



National  
Defence

Défense  
nationale



# **GROUND TERMINAL SIMULATOR IMPLEMENTATION FOR UPLINK SYNCHRONIZATION TRIALS**

by

**Caroline Tom**

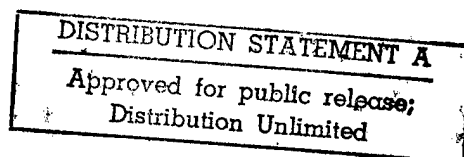
19990127 043

**DEFENCE RESEARCH ESTABLISHMENT OTTAWA**  
REPORT NO. 1341

**Canada**

November 1998  
Ottawa

DTIC QUALITY INSPECTED 1



AQF99-04-0762



National  
Defence

Défense  
nationale

# **GROUND TERMINAL SIMULATOR IMPLEMENTATION FOR UPLINK SYNCHRONIZATION TRIALS**

by

**Caroline Tom**

*Military Satellite Communications Group  
Space Systems and Technology Section*

**DEFENCE RESEARCH ESTABLISHMENT OTTAWA**  
REPORT NO. 1341

PROJECT  
5CA11

November 1998  
Ottawa

## **Abstract**

A ground terminal (GT) simulator was developed at Defence Research Establishment Ottawa (DREO) as part of an in-house activity examining aspects of uplink synchronization for extremely high frequency (EHF) satellite communications (SATCOM) using frequency hopping. The GT simulator consists of a GT processor, custom interface boards, synthesizer controller, frequency synthesizer, and data source. The GT processor is the principal component of the simulator and is realized by a TMS320C30 digital signal processor board. This report describes the implementation of the GT processor functions relating to uplink synchronization and the interfaces between the various components of the simulator. This report also describes the synchronization procedure for the GT simulator. The procedure is broken down into three steps: downlink synchronization; uplink coarse synchronization; and uplink fine synchronization. A guide on the hardware installation of the various components of the GT simulator and a list of the software needed to run the simulator is provided in an appendix.

## **Résumé**

Un simulateur de terminal au sol a été développé au Centre de Recherches pour la Défense à Ottawa (CRDO). Ce simulateur fait partie d'un travail au CRDO concernant les aspects de synchronisation de la liaison sol-espace pour les communications par satellite, utilisant le spectre étalé dans la bande de fréquence extrêmement haute. Le simulateur consiste d'un processeur de terminal au sol, des cartes d'interfaces fabriquées sur demande, un contrôleur de synthétiseur, un synthétiseur de fréquences, et une source de données. Le processeur du terminal au sol est la pièce principale du simulateur. Le rôle de processeur du terminal au sol est réalisé par une unité de traitement de signaux numériques, TMS320C30. Ce rapport décrit la réalisation des fonctions du processeur de terminal au sol liées à la synchronisation de la liaison sol-espace. Ce rapport décrit aussi la procédure pour la synchronisation du simulateur de terminal au sol. La procédure est divisée en 3 étapes: la synchronisation de la liaison espace-sol; la synchronisation préliminaire de la liaison sol-espace; et la synchronisation précise de la liaison sol-espace. Un guide d'installation des pièces du simulateur de terminal au sol est fourni ainsi qu'une liste des logiciels requis pour le fonctionnement du simulateur.

## Executive Summary

The Military Satellite Communications (MSC) Groups at Defence Research Establishment Ottawa (DREO) and Communications Research Centre (CRC) have been examining synchronization aspects of robust, anti-jam satellite communications at extremely high frequency (EHF). The MSC groups at DREO and CRC cooperatively developed ground terminal (GT) and payload simulators to carry out synchronization trials over the United Kingdom (UK) Skynet 4A EHF transponder. The use of the Skynet 4A EHF transponder was made possible through a Memorandum of Understanding established under The Technical Cooperation Program (TTCP). With the Skynet 4A EHF transponder, the ground-based GT and payload simulators are set up to realize a single path (either uplink or downlink) of a practical EHF communications link.

Before communications of a frequency hopped system can begin, synchronization of the GT and payload must be performed. The first step in the synchronization procedure is downlink synchronization, followed by uplink coarse synchronization, and finally uplink fine synchronization. The in-house activity examined downlink and uplink synchronization separately. The first part of the in-house activity focussed on downlink synchronization aspects and has been documented. During downlink synchronization, the GT gathers information about the system clock to allow it to proceed to uplink synchronization. In uplink synchronization, the GT attempts to align its clock with the payload by transmitting synchronization probes at designated times. The payload receives the probes and formulates synchronization responses. The responses are transmitted back to the GT and are used to adjust the GT clock.

This report describes the development of a GT simulator for the uplink synchronization trials. The GT simulator consists of a GT processor, a number of custom interface boards, frequency synthesizer, RF equipment, and a data source. The GT processor was implemented on a TMS320C30 digital signal processor (DSP) board and is contained in a host PC. The custom interface boards include a GT processor interface board which generates the necessary clock signals for the GT processor and provides the interface between the GT processor and the hopping synthesizer controller (HSC). The HSC, in turn, controls a frequency synthesizer. A multipurpose data interface board was also designed and fabricated to provide the interface between the GT processor and a data source, and to provide the interface to a downlink synchronization reference link.

The modes of operation for the GT simulator include: transmitting a continuous-wave (CW) tone at specific points in the hopping bandwidth; transmitting an arbitrary CW tone within the hopping bandwidth; sweeping a CW tone across the hopping bandwidth; performing downlink synchronization; performing uplink coarse synchronization; and performing uplink fine synchronization. The first three functions were implemented during the development of the simulator and are retained for debug purposes. The GT simulator has been developed so that commands for the various modes of operation can be issued remotely once the GT simulator is powered on and the executable file is run. In this implementation, remote operation of the GT

simulator is carried out by the payload simulator. The remote operation capability is included to facilitate the system integration and trials since the GT and payload simulators are physically located 1.5 km apart.

Although the focus of the second part of the in-house activity is only on uplink synchronization, there is a need to provide a mechanism for establishing downlink synchronization beforehand. As the experimental setup for the uplink synchronization only realizes a single path of an EHF communications link, a simulated downlink was set up between the GT and payload simulators using an RS232 serial communications link. During downlink synchronization, a downlink synchronization reference pulse is continuously transmitted by the payload to the GT simulator. The edges of the reference pulse correspond to specific instances in the pseudorandom hopping sequence of the payload simulator. The GT simulator detects and uses the pulse to form a preliminary estimate of the system clock.

Once downlink synchronization is achieved, the GT simulator can proceed with uplink coarse synchronization. In coarse synchronization, two consecutive bursts of sixteen synchronization probes are transmitted by the GT simulator using different timing hypotheses. In this implementation, an "outward moving" search scheme is used to test different timing hypotheses. The search scheme consists of starting at the most probable hypothesis obtained during downlink synchronization and shifting each subsequent timing hypothesis outward on either side of the most probable hypothesis. The payload receives the coarse synchronization probes and formulates a binary "detect/no detect" response. The response is relayed back to the GT simulator via another serial link. When a "detect" is received by the GT simulator, a verification process is carried out to ensure it is a valid "detect" response. Coarse synchronization is considered to be achieved when the GT clock is aligned to within a hop of the payload clock.

In fine synchronization, a single burst of thirty-two synchronization probes is transmitted at designated times. Again, the payload simulator receives the probes and computes a synchronization estimate to be returned to the GT simulator. The synchronization estimate represents how early or late the received probes are relative to the system clock. In order to reduce the number of times the GT clock is adjusted and to minimize any estimate errors due to noise, an average of ten fine synchronization responses is used to determine the final adjustment of the GT clock. The fine synchronization routine is repeated until the GT clock is aligned to within 10% of a hop of the payload clock.

The procedures for setting up the GT simulator are included in Appendix A. The listings of programs used by the GT simulator and the simulation parameter data files are contained in Appendix B and Appendix C respectively.

# Table of Contents

	<u>Page</u>
<b>ABSTRACT .....</b>	iii
<b>RÉSUMÉ .....</b>	iii
<b>EXECUTIVE SUMMARY .....</b>	v
<b>TABLE OF CONTENTS .....</b>	vii
<b>LIST OF FIGURES .....</b>	xi
<b>LIST OF TABLES .....</b>	xiii
<b>LIST OF SYMBOLS AND ABBREVIATIONS .....</b>	xv
 <b>1.0 Introduction .....</b>	 1
1.1 Background .....	1
1.2 Task Description .....	2
1.3 Report Outline .....	2
 <b>2.0 Synchronization Procedure .....</b>	 5
2.1 Downlink Synchronization .....	5
2.2 Uplink Synchronization .....	6
2.2.1 Uplink Coarse Synchronization .....	7
2.2.2 Uplink Fine Synchronization .....	8
2.2.3 Synchronization Responses .....	9
 <b>3.0 System Description .....</b>	 11
3.1 Simulator Setup .....	11
3.2 Ground Terminal Simulator Subsystem Hardware .....	12
3.2.1 Ground Terminal Processor .....	12
3.2.2 Ground Terminal Processor Interface Board .....	13
3.2.3 Hopping Synthesizer Controller and Frequency Synthesizer .....	13
3.2.4 Data Device and Multipurpose Data Interface Board .....	13
3.2.5 Synchronization Response Return Serial Link .....	13
3.2.6 Downlink Synchronization Reference Serial Link .....	14
3.2.7 Hardware Interface Requirements of the GT Simulator .....	14
3.2.7.1 DSPLINK Backplane Interface .....	14
3.2.7.2 Serial Link Interfaces .....	16
3.2.7.3 Data Source Interface .....	17
3.2.7.4 HSC Command and Transmit Data Interface .....	17
3.2.7.5 Frequency Synthesizer Interface for the HSC .....	17

## Table of Contents

	<u>Page</u>
3.3 Ground Terminal Simulator Software .....	17
3.3.1 DSP Assembly Language Programs .....	18
3.3.1.1 Main Assembly Language Program .....	18
3.3.1.1.1 Preliminary Initialization by the GT Processor DSP .....	18
3.3.1.1.2 GT Simulator Modes of Operation .....	19
3.3.1.2 Coarse Synchronization Assembly Routine .....	20
3.3.1.2.1 Starting the Coarse Synchronization Procedure .....	20
3.3.1.2.2 Coarse Synchronization Probe Generation .....	21
3.3.1.2.3 Verification of "Detect" Responses Received .....	23
3.3.1.2.4 Preparing for Fine Synchronization .....	27
3.3.1.2.5 Clearing the Synchronization Response Buffer .....	28
3.3.1.2.6 Search Range for Coarse Synchronization Routine Exceeded .....	28
3.3.1.2.7 Synchronization Response Buffer Overflow .....	28
3.3.1.3 Fine Synchronization Assembly Routine .....	28
3.3.1.3.1 Beginning Fine Synchronization .....	29
3.3.1.3.2 Generation of Fine Synchronization Probes .....	29
3.3.1.3.3 Refining the Alignment of the Ground Terminal Clock .....	30
3.3.1.3.4 Nonconvergence of Fine Synchronization Estimates .....	31
3.3.1.3.5 Achieving Fine Synchronization .....	31
3.3.1.4 Interrupt Service Routine .....	31
3.3.1.4.1 ISR Housekeeping .....	31
3.3.1.4.2 ISR Tasks for Coarse Synchronization Mode .....	32
3.3.1.4.3 ISR Tasks for Fine Synchronization Mode .....	33
3.3.1.4.4 ISR Tasks for Downlink Synchronization Mode .....	33
3.3.2 Host/User Interface Program .....	34
3.3.2.1 DSP Board Initialization .....	34
3.3.2.2 Downloading Simulation Parameters .....	34
3.3.2.3 Serial Communications Initialization .....	35
3.3.2.4 User Interface Menu .....	35
3.3.2.5 Host/User Interface Loop .....	36
3.3.2.5.1 Checking for User Input - Local Keyboard Input .....	37
3.3.2.5.2 Checking for User Input - Remote Input .....	37
3.3.2.5.3 Checking for FR0 Pulse Edge Detection .....	38
3.3.2.5.4 Checking for Completion of Task by DSP .....	38
3.3.2.5.5 Checking for Return Link Synchronization Response .....	38
3.3.2.5.6 Checking Other Flag Conditions .....	38
3.3.3 ASCII Data Files .....	39
 4.0 Summary .....	 41
 References .....	 43

## Table of Contents

	<u>Page</u>
<b>Appendix A - Ground Terminal Simulator Installation Guide</b>	
A1 Installation .....	A1
A1.1 Hardware Installation .....	A1
A1.2 Program Files .....	A2
<b>Appendix B - Software Listings</b>	
B1 GT Simulator Host/User Interface Program .....	B1
B2 DSP Main Program .....	B17
B3 Coarse Synchronization Routine .....	B33
B4 Fine Synchronization Routine .....	B50
B5 DSP Interrupt Service Routine .....	B57
B6 TMS Linker .cmd File .....	B62
<b>Appendix C - ASCII Data Files</b>	
C1 General .....	C1
C2 Freq.dat File .....	C1
C3 Hscinit.dat File .....	C2
C4 Gtparam.dat File .....	C3



## List of Figures

	<u>Page</u>
Fig. 2.1	Downlink transmission structure 5
Fig. 2.2	Downlink synchronization reference pulse detail 6
Fig. 2.3	General time-frequency plan for uplink transmission structure 7
Fig. 2.4	Data flow of the coarse synchronization process 8
Fig. 2.5	Data flow of the fine synchronization process 9
Fig. 2.6	Synchronization response format 10
Fig. 3.1	System block diagram of the uplink synchronization experiments 12
Fig. 3.2	DSPLINK backplane interface connector 15
Fig. 3.3	Pinout configuration for serial connector to downlink synchronization reference pulse 16
Fig. 3.4	Coarse synchronization procedure state diagram 20
Fig. 3.5	Search scheme for coarse synchronization timing hypotheses 22
Fig. 3.6	Different scenarios for GT clock alignment with payload clock 25
Fig. 3.7	Fine synchronization procedure state diagram 29
Fig. 3.8	Interrupt service routine flow diagram 32
Fig. 3.9	Host/user interface flow diagram 34
Fig. 3.10	User menu for the GT simulator 35
Fig. 3.11	Host/user interface loop operation 37
Fig. A.1	GT simulator TMS320C30 linker process A2
Fig. A.2	GT simulator host PC linker process A3

## List of Tables

		<b><u>Page</u></b>
Table 3.1	DSPLINK backplane interface pinout description	16
Table 3.2	GT simulator modes of operation	19
Table 3.3	Description of flags for GT simulator	40

## List of Symbols and Abbreviations

<b>ADJ_NCO</b>	Adjust NCO state in fine synchronization routine
<b>ASCII</b>	American Standard Code for Information Interchange
<b>CLR_RESP_PIPE</b>	Clear Response Buffer Pipeline state in coarse synchronization routine
<b>CRC</b>	Communications Research Centre
<b>CSYNC</b>	Coarse Synchronization mode command of the HSC
<b>CW</b>	Continuous Wave
<b>DIB</b>	Data Interface Board
<b>DP</b>	Data Page
<b>DREO</b>	Defence Research Establishment Ottawa
<b>DSP</b>	Digital Signal Processor
<b>EHF</b>	Extremely High Frequency
<b>FCALC</b>	Frequency Calculate command for the HSC
<b>FFT</b>	Fast Fourier Transform
<b>FIFO</b>	First In First Out
<b>FINE_NT_ACH</b>	Fine Synchronization Not Achieved state in fine synchronization routine
<b>FR0</b>	Frame 0
<b>FSK</b>	Frequency Shift Keying
<b>FSK/FRAME</b>	Transmit data port of the GT processor i/f board
<b>GEN_PROBES</b>	Generate Probes state in coarse synchronization routine
<b>GO_2_FSYNC</b>	Go to Fine Synchronization state in coarse synchronization routine
<b>GO_2_RUN</b>	Go to RUN state in fine synchronization routine
<b>GT</b>	Ground Terminal
<b>HSC</b>	Hopping Synthesizer Controller
<b>i/f</b>	Interface
<b>I/O</b>	Input/Output
<b>INIT_SECTION</b>	Initialization Section in fine synchronization routine
<b>ISR</b>	Interrupt Service Routine
<b>JDLS</b>	Joint Data Link Standard
<b>MOU</b>	Memorandum Of Understanding
<b>MSC</b>	Military Satellite Communications
<b>NCO</b>	Numerically-Controlled Oscillator
<b>PC</b>	Personal Computer
<b>PL</b>	Payload
<b>PLINE_ERR</b>	Pipeline Overflow Error state in coarse synchronization routine
<b>PRELIM_INIT</b>	Preliminary Initialization state in coarse synchronization routine
<b>proc</b>	Processing
<b>prop</b>	Propagation
<b>RF</b>	Radio Frequency
<b>SATCOM</b>	Satellite Communications
<b>SHF</b>	Super High Frequency
<b>SRCH_EXCEED</b>	Search Range Exceeded state in coarse synchronization routine
<b>TTCP</b>	The Technical Cooperation Program
<b>TXOFF</b>	Transmit off command bit for the HSC
<b>TX_FPROBES</b>	Transmit Fine Synchronization Probes in fine synchronization routine

## **List of Symbols and Abbreviations**

<b>UK</b>	United Kingdom
<b>ULGO</b>	Uplink GO command for the HSC
<b>VER_DETECT</b>	Verify Detect state in coarse synchronization routine

## **1.0 Introduction**

### **1.1 Background**

The Military Satellite Communications (MSC) Group at Defence Research Establishment Ottawa (DREO), along with its sister group at Communications Research Centre (CRC) have been examining aspects of robust, anti-jam satellite communications at extremely high frequency (EHF). An area critical to the operation of EHF frequency hopped satellite communications (SATCOM) systems is the synchronization of the ground terminal (GT) clock to the payload system clock. In order to gain a better appreciation of the processes involved in synchronization, the MSC groups at DREO and CRC developed GT and payload simulators to carry out synchronization trials. Trials were carried out over the United Kingdom (UK) Skynet 4A EHF transponder, made possible through a memorandum of understanding (MOU) established under The Technical Cooperation Program (TTCP). The Skynet 4A transponder receives an EHF signal and translates it to a super high frequency (SHF) signal which is subsequently retransmitted. Consequently, the ground-based simulators for the GT and the payload are used to realize a single path of a practical EHF communications link.

Before communications of an EHF system can begin, synchronization of the GT and payload must be performed. Downlink synchronization is considered to be the first step in the synchronization process and was the focus of the first part of the in-house activity examining EHF satellite communications. The work on downlink synchronization was documented in [1]. During downlink synchronization, the GT gathers synchronization information about the system clock to allow it to receive frequency hopped signals from the satellite. The synchronization information also provides a basis to begin uplink synchronization. The GT simulator performs uplink synchronization in order to align its clock with the payload clock after taking into account the non-deterministic propagation delay. Processing delays, although small by comparison, can also be factored into the alignment of the GT clock. Once uplink synchronization is achieved, the GT simulator can begin to transmit user data over the frequency hopped SATCOM system.

The uplink synchronization process consists of two phases: coarse synchronization and fine synchronization. In both phases, the GT simulator transmits synchronization probes to the payload. The payload receives the probes and formulates a synchronization response for the GT simulator. The synchronization response during coarse synchronization indicates whether the probes are detected by the payload simulator. In fine synchronization, the synchronization response reflects how early or late the received probes are, relative to the payload clock. The synchronization responses are relayed to the GT simulator and are used by the GT simulator to adjust its clock. The aspects of uplink synchronization were the focus of the second part of the in-house activity and are described in this report.

Users of an EHF satcom system communicate according to a data link standard which defines the uplink and downlink signal formats and the processing operations. For the in-house activity, a Joint Data Link Standard (JDLS) was written [2] to provide the parameters for uplink

and downlink transmission structures. The parameters include the modulation scheme, data rate, frame and channel sizes, and subframe allocations.

## **1.2 Task Description**

In order to carry out uplink synchronization experiments, GT and payload simulators had to be developed. Each of the simulators consists of a processing unit which performs the synchronization tasks and interfaces to other components of the simulator. The payload processor must receive, demodulate, and process transmitted synchronization probes. The payload processor must also formulate responses corresponding to the synchronization probes received and transmit the responses back to the GT simulator. In addition, the payload processor must generate a synchronization aid which is used by the GT simulator to establish a starting point for performing uplink synchronization. The generation of a synchronization aid is intended as a substitute for the downlink synchronization process described above since only the uplink path can be realized using the transponding Skynet satellite. For the purposes of the in-house activity and of this report, the detection of the synchronization aid by the GT simulator is referred to as performing downlink synchronization. The payload processor also interacts with other hardware components of the simulator. The hardware components include a hopping synthesizer controller (HSC) which controls a frequency synthesizer, a data interface board which connects to a data sink, and a first-in-first-out (FIFO) interface board which holds the samples of the received signal.

The GT processor performs reciprocal tasks of the payload processor. The GT processor assembles and transmits modulated uplink synchronization probes at specific allocated times according to the data link standard. The GT processor must also receive and decode the synchronization responses during uplink synchronization. As well, the GT processor is responsible for detecting the synchronization aid generated by the payload simulator. Furthermore, the GT processor interacts with other components of the GT simulator. Once such component is a GT processor interface (i/f) board which generates the necessary GT clock signals and transfers commands to an HSC. The HSC, in turn, controls a transmitting frequency synthesizer. The GT processor also communicates with a data interface board which is connected to the data source.

## **1.3 Report Outline**

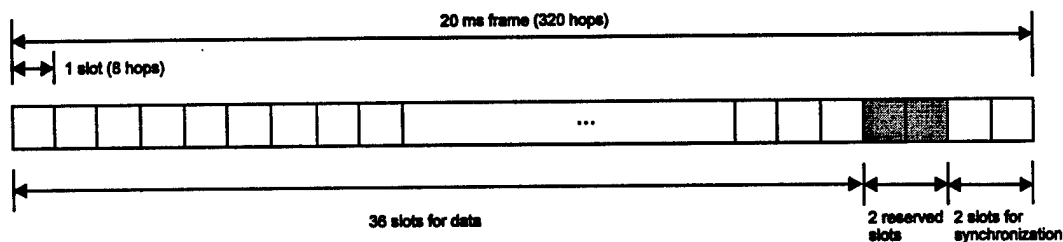
The purpose of this report is to describe the development of the GT simulator for the uplink synchronization experiments. A detailed description of the synchronization procedure is provided in Section 2.0. In Section 3.0, the simulator setup is described to introduce the components of the GT simulator. Subsequently, the description of the GT simulator is broken down into the hardware and software features of the simulator. The hardware section deals mainly with the physical components and interfaces. The software section describes the tasks performed by the GT simulator. Three appendices are included in this report. Appendix A consists of a guide which outlines the installation and setup procedures of the GT simulator.

Appendix B contains a listing of all the programs used for the GT simulator. Appendix C contains a copy of the American Standard Code for Information Interchange (ASCII) data files used by the GT simulator to download specific simulation parameters.

## 2.0 Synchronization Procedure

### 2.1 Downlink Synchronization

The first step in establishing communications over an EHF frequency hopped SATCOM system consists of achieving downlink synchronization. The purpose of performing downlink synchronization is to acquire the satellite downlink including demodulating user data and synchronization information. The downlink synchronization process preceeds uplink synchronization. From the data link standard described in [2], the payload terminal transmits synchronization hops based on a time division multiplex scheme. The GT simulator, upon receiving and detecting the synchronizataion hops, is then able to derive an estimate of the payload (system) clock in order to begin uplink synchronization. The general downlink transmit structure is shown in Fig. 2.1. A 20 ms frame consisting of 320 hops is further divided into 40 time slots of 8 hops each. The data link standard specifies that the first 36 time slots are allocated to user data. The next two are reserved and the last two are used for synchronization purposes.

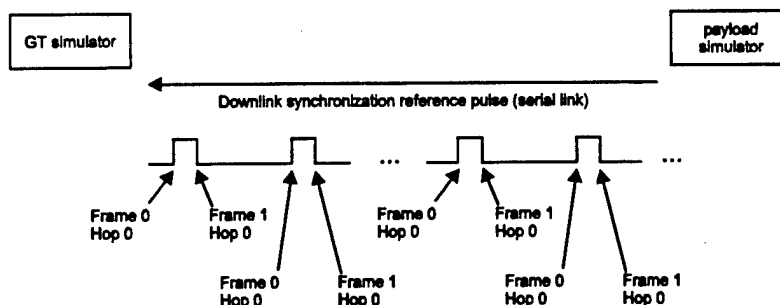


**Fig. 2.1 Downlink transmission structure**

As described in Section 1.1, the uplink synchronization trials are carried over the Skynet 4A EHF transponder. As a transponding satellite is used, only one path (uplink or downlink) of an EHF satcom system with onboard processing can be realized at any one time. The use of a transponding satellite for the uplink synchronization trials also means both the GT simulator and payload simulator are developed as ground-based systems. For the uplink synchronization trials, a downlink "path" is simulated by a direct serial link connection between the payload and GT simulators. Furthermore, to facilitate the implementation of the simulated downlink, a reference pulse is transmitted in lieu of synchronization hops to transmit a reference of the payload terminal clock (master clock). The reference pulse is referred to as the downlink synchronization reference pulse and is shown in Fig. 2.2. The edges of the reference pulse were chosen to correspond to specific points in the pseudorandom hop sequence. The rising edge of the downlink synchronization reference pulse corresponds to the start of hop number 0 of frame number 0 in the pseudorandom sequence. The falling edge of the reference pulse corresponds to the start of hop number 0 of frame number 1. The GT simulator receives this reference pulse and resets its own hop clock accordingly in preparation for uplink synchronization. For the uplink



synchronization experiments, the downlink synchronization reference pulse is continuously generated and transmitted by the payload simulator.

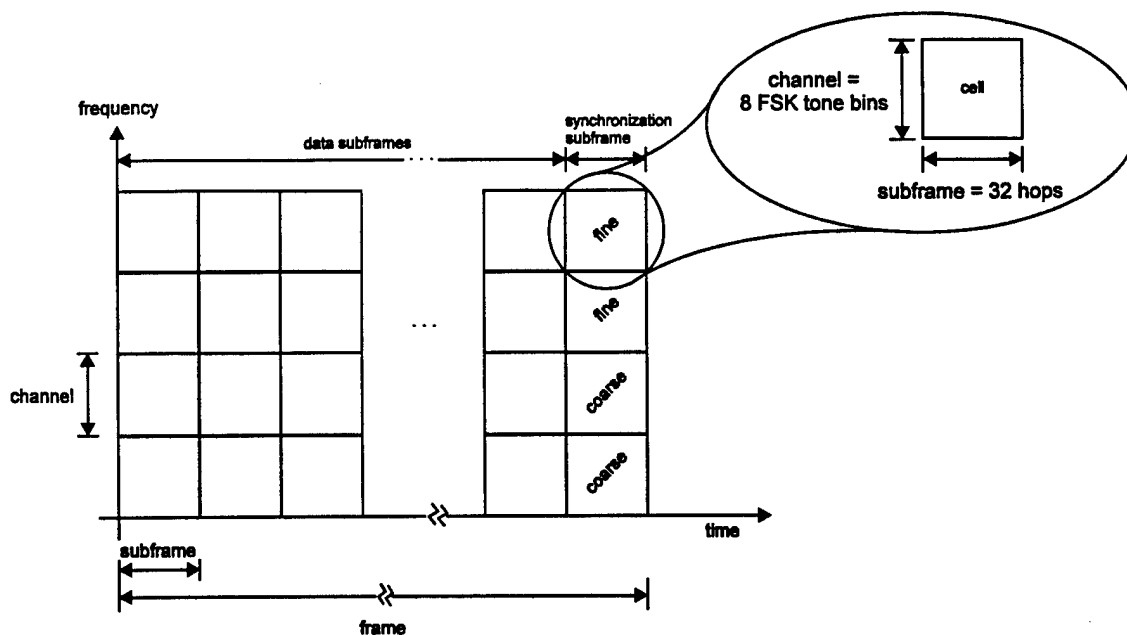


**Fig. 2.2 Downlink synchronization reference pulse detail**

## 2.2 Uplink Synchronization

Once downlink synchronization is achieved, the GT simulator can proceed with uplink synchronization. Uplink synchronization is the process whereby the GT simulator attempts to align its own clock with the payload (system) clock. The uplink synchronization process is carried out independently from downlink synchronization because in uplink synchronization, the ground terminal takes into account the propagation delay so that its transmissions are received at the appropriate time by the payload. The GT simulator transmits bursts of synchronization probes which correspond to different timing offset hypotheses. The payload simulator receives the probes and produces a response. The response indicates the coarse synchronization probes detection and the fine synchronization probe timing offset relative to the payload clock in a particular frame. The GT simulator then uses the responses to align its clock with the payload clock.

The timing offset hypotheses of the transmitted synchronization probes correspond to specific instances relative to the time-frequency plan described in [2] and illustrated in Fig. 2.3. The format of the time-frequency plan is based on a multichannel subframe/frame structure which is repeated on a frame basis. A subframe for a particular channel is referred to as a cell [2]. There are four channels implemented for the uplink synchronization trials. Each channel is subdivided into 8 frequency tone bins to support the 8-ary frequency-shift-keying (FSK) uplink modulation scheme specified in [2]. Each cell is designated as either a data cell or a synchronization cell for transmission of data or synchronization probes respectively. A particular user would be assigned specific cells to transmit data or synchronization probes.



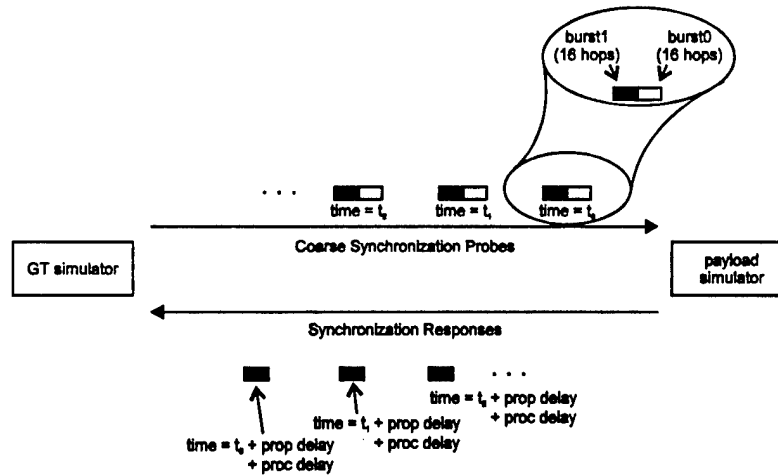
**Fig. 2.3 General time-frequency plan for uplink transmission structure**

The uplink synchronization process is carried out in two stages. The first stage is referred to as coarse synchronization. Coarse synchronization is achieved when the payload detects the GT simulator's synchronization probes. For the uplink synchronization trials, coarse synchronization occurs when the GT simulator clock is within a hop of the payload clock. The second stage of uplink synchronization is called fine synchronization. During fine synchronization, the GT simulator attempts to refine the alignment of its clock to come within 10% of a hop. The method for carrying out coarse and fine synchronization are described further in the following subsections.

### 2.2.1 Uplink Coarse Synchronization

Coarse synchronization probes are transmitted in a specific cell in the time-frequency plan and are composed of two contiguous bursts of sixteen probes [2]. The synchronization probes are located in a specific FSK tone bin. The probes are subsequently frequency hopped for transmission. In this implementation, the coarse synchronization probes are generated by a frequency synthesizer which is controlled by the HSC [3]. Each sixteen probe burst has a different timing hypothesis. The procedure for generating the coarse synchronization probes is given in Section 3.3.1.2.2. The GT simulator transmits the coarse synchronization probes by issuing a command to the HSC to precompute the frequency hopped probe frequencies for the two bursts and by commanding the HSC to switch to those frequencies at an appropriate time. After a propagation delay and a processing delay, a binary response is returned by the payload simulator as a synchronization response indicating if the coarse synchronization probes were detected. Details of the synchronization response are provided in Section 2.2.4. The consistent detection of coarse synchronization probes with the same hypothesis indicates that the GT clock

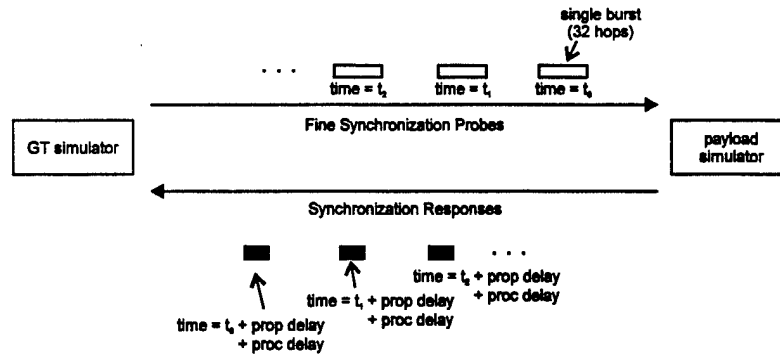
is within a hop of the payload clock. Coarse synchronization is thus achieved. The general data flow for coarse synchronization is shown in Fig. 2.4.



**Fig. 2.4 Data flow of the coarse synchronization process**

### 2.2.2 Uplink Fine Synchronization

Fine synchronization probes are also transmitted in a specific FSK tone bin of a specific cell in the time-frequency plan. However, fine synchronization probes consist of a single burst of 32 probes rather than two bursts of 16 as in the coarse synchronization process. Again the synchronization probes are generated by a frequency synthesizer. Information on the specific cell (i.e. channel number and FSK tone bin) to be used for the fine synchronization probes is transmitted by the GT processor to the HSC at the appropriate time (subframe). The payload simulator processes the received probes and formulates a response which indicates how early or late the probes are in relation to the payload clock. Details of the synchronization response format are given in Section 2.2.4. For the uplink synchronization experiments, uplink synchronization is considered achieved when the GT clock is aligned to within 10 % of the payload clock. The value of 10% was considered reasonable to account for frequency drift while causing minimal degradation to FSK modulation. A general data flow diagram for the fine synchronization process is shown in Fig. 2.5.



**Fig. 2.5 Data flow of the fine synchronization process**

### 2.2.3 Synchronization Responses

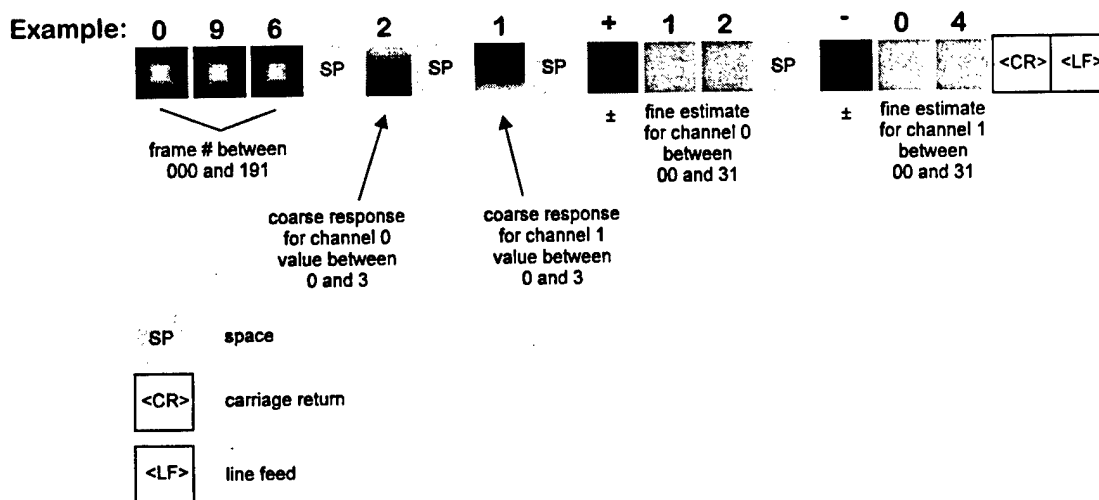
Synchronization responses are formulated by the payload simulator each frame to provide feedback to the GT simulator on the detection of coarse synchronization probes and on the estimation of the time offset of fine synchronization probes. For the uplink synchronization experiments, a synchronization response is returned via a synchronization response return link. The return link is described in Section 3.2.5. The synchronization response return link is implemented using an RS232 serial connection between the GT and payload. The serial connection is capable of supporting communications at 9.6 kb/s. With one response being transmitted per 20ms frame, the serial communications link is able to support synchronization responses which are about 20 characters in length.

For the uplink synchronization trials, the synchronization response consists of a reference frame number, the coarse synchronization detection results for the two coarse synchronization probe channels, and the fine synchronization estimates for the the two fine synchronization probe channels. The frame number ranges in value between 0 and 191. The coarse synchronization response for each synchronization burst is a binary response, i.e. it is either a “detect” or “no detect”. The “detect” and “no detect” responses are represented by “1” and “0” respectively. As there are two probe bursts per channel per frame, there are four possible combinations of detection responses for each channel. In this implementation, the detection result for the first probe burst is selected to occupy the most significant bit. The combinations of detection results are mapped to a decimal representation for the return link synchronization response and thus, are represented by a value between 0 and 3.

The fine synchronization response is a number between -31 and 31 representing the timing error of the GT clock. A negative fine synchronization response indicates that the GT clock is early while a positive fine synchronization estimate indicates that the GT clock is late. It is assumed that coarse synchronization is achieved prior to performing fine synchronization so that the GT clock is within a hop of the payload clock at the start of fine synchronization. As a

result, with a hop period of 62.5  $\mu$ s, the fine synchronization estimate (timing error) represents approximately half the actual timing error, in  $\mu$ s.

In order to comply with the message length restrictions for the serial connection, it was decided that the synchronization response would be formatted as shown in Fig. 2.6. An example of a synchronization response received by the GT simulator is also included.



**Fig. 2.6 Synchronization response format**

For the example in Fig. 2.6, the GT simulator would subsequently decode the response as:

- 096 ➡ Synchronization response for frame number 96
- 2 ➡ Coarse synchronization response for channel 0  
 $2_{10} = 10_2$  "detect" for burst 0  
"no detect" for burst 1
- 1 ➡ Coarse synchronization response for channel 1  
 $1_{10} = 01_2$  "no detect" for burst 0  
"detect" for burst 1
- +12 ➡ Fine synchronization response for channel 0,  
payload estimates that received probes are 24  $\mu$ s  
later than payload clock
- 04 ➡ Fine synchronization response for channel 1,  
payload estimates that received probes are 8  $\mu$ s  
earlier than payload clock

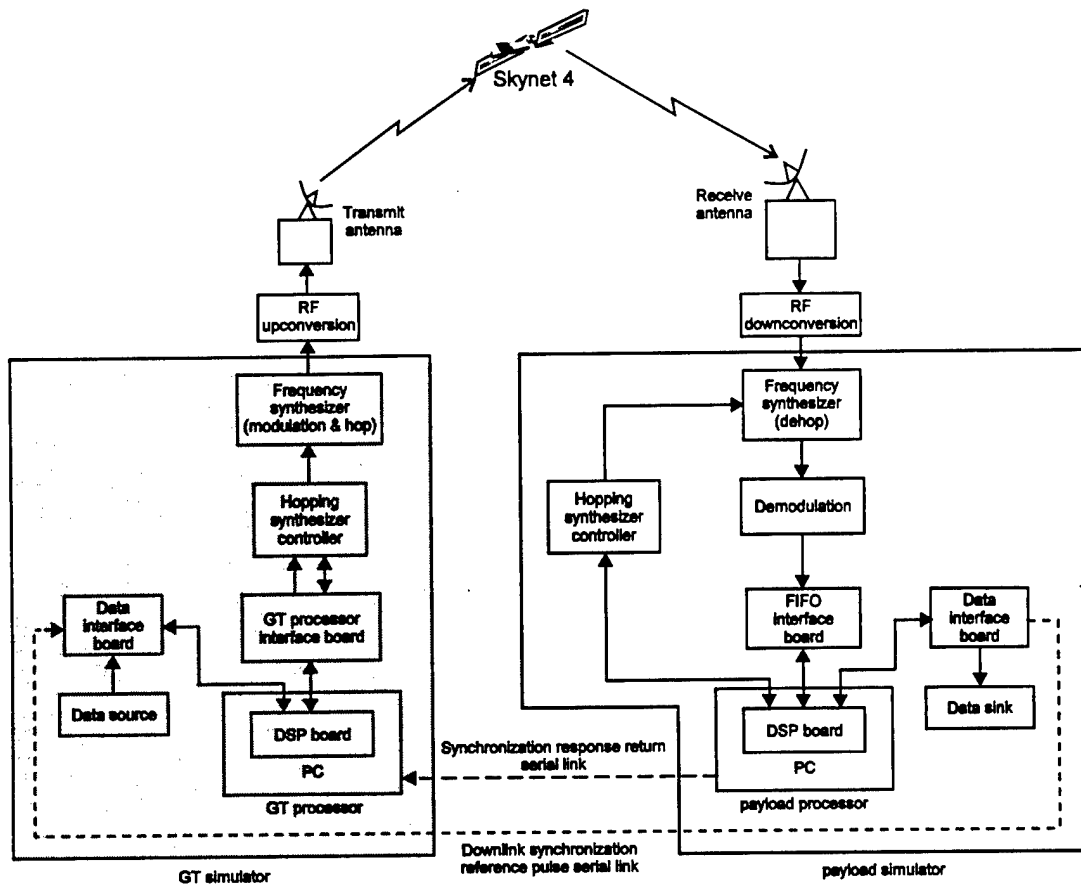
## 3.0 System Description

### 3.1 Simulator Setup

A system block diagram of the simulator setup for the uplink synchronization experiments is shown in Fig. 3.1. The GT and payload simulators are ground-based systems which are located approximately 1.5 km apart on the DREO/CRC site.

Data to be transmitted from the GT is modulated using 8-ary FSK as specified in [2]. The GT processor transfers the modulated data to the GT processor i/f board [4]. The data is read by the HSC which calculates the frequency of the next hop. The result of the calculations is a frequency value which includes the modulated data and the hop frequency. The HSC then passes this frequency value to the frequency synthesizer. The frequency synthesizer output is then converted to the radio frequency (RF) transmit signal at EHF. Upon receiving the EHF transmitted signal, Skynet 4 translates and retransmits the signal at X-band to the payload terminal. The received signal is downconverted and transferred to a frequency synthesizer which is controlled by another HSC. The frequency synthesizer generates the hopping pattern used to dehop the received signal. A Fast-Fourier Transform (FFT)-based processor is then used to produce samples of the received signal for each FFT channel [5]. The samples are stored on a FIFO interface board until the payload processor is ready to process the data.

The entire process described above encompasses only the uplink portion of an actual EHF satcom system with on-board processing. In an actual system, when the payload receives and processes the data during synchronization, a response is formulated and transmitted back to the GT on the downlink. For the uplink synchronization trials, the downlink portion is simulated using a direct serial link called the "synchronization response return link" which will be used to transmit the responses generated by the payload processor to the GT. The synchronization response return link is separate from the downlink synchronization reference link described in Section 2.1 which is used to simulate downlink synchronization. Furthermore, the interface of the synchronization response return link to each simulator is through the host personal computer (PC). By contrast, the interface of the downlink synchronization reference pulse serial link to each simulator is provided through a multipurpose data interface board designed and fabricated at DREO [6]. A data source and data sink are included in the simulator setup for data communications once fine synchronization is achieved.



**Fig. 3.1 System block diagram of the uplink synchronization experiments**

## 3.2 Ground Terminal Simulator Hardware

### 3.2.1 Ground Terminal Processor

The GT processor for the uplink synchronization experiments consists of a Texas Instruments' TMS320C30 digital signal processor (DSP) board. The TMS320C30 board is installed in a single 16-bit slot of an IBM-compatible host PC with a monitor and keyboard. Communications between the PC and the DSP board go through the PC's input/output (I/O) space. The GT processor communicates with other components of the GT simulator using the serial port of the PC or by using the DSPLINK interface supported by the TMS320C30 DSP board. These interfaces are discussed further in the subsections below.

### 3.2.2 Ground Terminal Processor Interface Board

A GT processor interface board was designed and fabricated at DREO for the uplink synchronization experiments and is documented in [4]. The board was designed to perform three functions: to generate the necessary clock signals for the GT simulator operations; to provide a command interface for a hopping synthesizer controller [3]; and to provide an interface for transferring FSK/channel data to the hopping synthesizer controller. Communications between the GT processor and the GT processor i/f board is achieved through the DSPLINK interface of the TMS320C30 DSP board. The GT processor i/f board is installed in a DSPLINK backplane chassis that was also assembled at DREO. The DSPLINK interface and backplane are described further in Section 3.2.7.1. Details on the specific operation of the GT processor i/f board can be found in [4].

### 3.2.3 Hopping Synthesizer Controller and Frequency Synthesizer

A frequency synthesizer is used in the uplink synchronization experiments to produce the appropriate frequency-hopped FSK tone to be transmitted by the GT. In this implementation of the GT simulator, two synthesizers are supported: the Comstron FS2000 Frequency Synthesizer [7]; and the Sciteq VDS-2G-469 Frequency Synthesizer [8]. Both frequency synthesizers are driven by an HSC which was developed at DREO [3]. The HSC, in turn, is controlled by the GT processor via the GT processor i/f board which is described above. The HSC receives and actions commands from the GT processor relating to its initialization and mode of operation. The HSC performs all the calculations required for frequency computation (including a random number generator routine for the pseudorandom hop sequence), and frequency word format conversion. Finally, the HSC transfers the resulting frequency word to the frequency synthesizer at the appropriate point in time. A detailed description of the HSC is provided by [3].

### 3.2.4 Data Device and Multipurpose Data Interface Board

The data source for the GT simulator subsystem consists of an HP1645A Bit-error-rate test set [9]. The interface between the HP1645A and the GT processor is realized using a multipurpose data interface board (DIB) which was developed at DREO and is documented in [6]. The DIB receives a single-bit RS232 data stream from the HP1645A, converts the data to TTL levels, and formats the data stream into 12-bit words to be read by the GT processor. The DIB is installed in the DSPLINK backplane chassis and communicates with the GT processor via the DSPLINK interface. The DSLINK interface and backplane are discussed in Section 3.2.7.1.

### 3.2.5 Synchronization Response Return Serial Link

As mentioned in Section 2.2.3, the payload simulator formulates a synchronization response for every frame. The synchronization response contains information on whether any



probes were detected in the coarse synchronization cells which are shown in Fig. 2.3. The synchronization response also contains estimates of the timing offset of probes which may be received in the fine synchronization cells. In a practical EHF satcom system with on-board processing, the synchronization response is transmitted by the payload to the GT on the downlink. For the uplink synchronization experiments, a transponding satellite is used and thus, another means of transmitting the synchronization responses to the GT simulator is required. A synchronization response return link is realized using an RS232 serial connection between the GT and payload ground-based simulators. The serial connection supports communications at 9.6 kb/s and is accessed through the serial connector of the PC host for each simulator. Serial communications software [10] developed at DREO is used by the payload and GT hosts to send and retrieve the synchronization responses respectively.

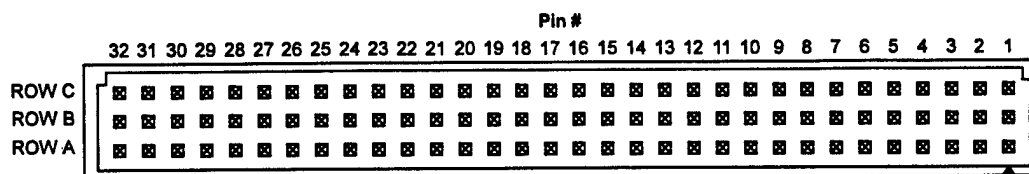
### 3.2.6 Downlink Synchronization Reference Serial Link

A downlink synchronization reference serial link is implemented to transmit the downlink synchronization reference pulse which is described in Section 2.1. The downlink synchronization reference pulse is analogous to using synchronization hops of a real processing satellite. Similar to the case for the relaying of synchronization responses back to the GT, another method of transmitting the reference pulse is required as a result of a transponding satellite being used for the trials. In order to be able to present the reference pulse to the GT processor in real time, it was decided to route the downlink synchronization reference serial link by way of the DIB. By using the DIB, it is possible to avoid having to use the slower host/DSP interface. Details of the downlink synchronization reference serial link implementation are found in [6].

### 3.2.7 Hardware Interface Requirements of the GT Simulator

#### 3.2.7.1 DSPLINK Backplane Interface

In order to support multiple custom boards using the DSPLINK interface to communicate with the GT processor, a DSPLINK backplane chassis was assembled for the uplink synchronization experiments. Currently, the backplane chassis houses the GT processor i/f board, the DIB, and an adaptor board to map the 50-pin DSPLINK interface [11] connector from the DSP board to the 96-pin backplane connector. The DSPLINK backplane connector and pinout description are shown in Fig. 3.2 and Table 3.1 respectively.



### Corresponding signal names

PIN	ROW A	ROW B	ROW C
1	HOP CLK		D0
2	HOP CLK*		D1
3			D2
4		RESERVED	D3
5		RESERVED	D4
6		RESERVED	D5
7		RESERVED	D6
8		RESERVED	D7
9	GND	RESERVED	GND
10	DATA CLK	RESERVED	D8
11	GND	RESERVED	D9
12			D10
13			D11
14			D12
15	GND		D13
16			D14
17	GND		D15
18			W*/R
19	GND		IOE*
20	RESERVED	GND	INTO*
21	RESERVED		RESET
22	RESERVED		CLK/2
23		GND	A0
24			A1
25			A2
26			A3
27			FLAGIN
28		-5V ANALOG	FLAGOUT
29		5V ANALOG	
30		AGND	
31	-15V	5V STBY	15V
32	15V	5V	5V

The asterisk (\*) denotes an active-low signal

**Fig. 3.2 DSPLINK backplane interface connector**

Signal	Direction with respect to DSP	Details
D0-D15	to/from	Sixteen bi-directional TTL data lines of DSPLINK
GND	-	Digital ground
W*/R	from	DSPLINK read/write* line to signal the direction of data transfer
IOE*	from	An active-low, input/output enable signal indicating an access on the DSPLINK
INT0*	to	A negative-edge triggered, or active-low interrupt signal on DSPLINK generated on the GT processor i/f board.
RESET*	from	DSPLINK reset line.
CLK/2	from	General purpose clock signal . This signal is not used by the GT processor i/f board.
A0-A3	from	Four buffered TTL address lines of DSPLINK.
FLAGIN	to	General purpose input line on DSPLINK readable by the DSP. This signal is not used by the GT processor i/f board.
FLAGOUT	from	General purpose output line on DSPLINK writeable by the DSP.
15V	-	15 volts power supply.
-15V	-	-15 volts power supply
5V	-	5 volts power supply
-5V ANALOG	-	-5 volts analog power supply
5V ANALOG	-	5 volts analog power supply
AGND	-	Analog ground
5V STDBY	-	5 volts standby power supply. This signal is not used by the GT processor i/f board.
HOP CLK	to	Hop clock signal originating from the GT processor i/f board.
HOP CLK*	to	Inverse hop clock signal. This pin is not currently being used by the GT processor i/f board.
DATA CLK	-	Data clock signal originating from the GT processor i/f board.
RESERVED	-	Reserved lines for the DSP backplane.

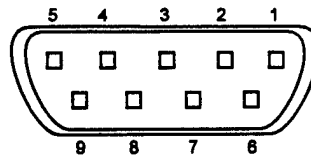
The asterisk (\*) denotes an active-low signal

**Table 3.1 DSP backplane interface pinout description**

### 3.2.7.2 Serial Link Interfaces

There are two serial link interfaces required for the GT and payload simulators. The first interface is provided by a standard serial connector located on the host PC to access the synchronization response return serial link. As mentioned in Section 3.2.5, serial communications software routines were developed and written at DREO [10] to provide the user interface to the synchronization response return link.

The second serial link is used to relay the downlink synchronization reference pulse to the GT simulator. The downlink synchronization reference pulse serial link interface is a 9-pin D-type connector located on the DIB. The pinout description of the D-type connector is shown in Fig. 3.3.



□ Female connector

<u>Pin</u>	<u>Description</u>
1	not used
2	FR0 in
3	FR0 out
4	not used
5	GND
6	not used
7	not used
8	not used
9	not used

**Fig. 3.3 Pinout configuration for serial connector to downlink synchronization reference pulse**

### 3.2.7.3 Data Source Interface

The connection between the data source and the DIB shown in Fig. 3.1 is realized by a male 9-pin D-type connector and is described further in [6].

### 3.2.7.4 HSC Command and Transmit Data Interface

As shown in Fig. 3.1, there are two connections between the GT processor i/f board and the HSC. These connections correspond to the HSC's command and transmit data interface. The connections are made using of a 26-pin ribbon cable and a 10-pin ribbon cable respectively. The details of the command and transmit data interface are described in [3] and [4].

### 3.2.7.5 Frequency Synthesizer Interface for the HSC

A 50-pin ribbon cable is used to connect the HSC to the frequency synthesizer. A description of the connection between HSC and the frequency synthesizer, as shown in Fig. 3.1, is given in [3].

## 3.3 Ground Terminal Simulator Software

The following section provides a description of the software for the DSP and the host. In both cases, the principal concepts and operations of the routines are outlined. The software for the DSP is described first. The DSP software includes the coarse and fine synchronization

routines and the processing of synchronization responses. Subsequently, the host/user interface software is described.

### 3.3.1 DSP Assembly Language Programs

The GT processor functions are implemented on a Texas Instruments' TMS320C30 DSP board which is contained in a PC. The software for the GT processor was written in assembly language and is described in the following subsections.

#### 3.3.1.1 Main Assembly Language Program

The main DSP program for the GT processor performs two general functions. The first function is the preliminary initialization of the DSP board, the interface boards, and associated hardware of the GT simulator. The second function of the main DSP program is to respond to commands issued by the host program. The commands correspond to different modes of operation for the GT simulator. The two functions of the main DSP program are described further in the following subparagraphs.

##### 3.3.1.1.1 Preliminary Initialization by the GT Processor DSP

The first stage of the main assembly program involves putting the DSP and its associated interface boards and hardware into a known state. The TMS320C30 DSP data page (DP) pointer, status register, stack pointer, primary bus control register, and secondary bus control register are initialized to appropriate values as specified in [11] and shown in Section B.3 of Appendix B. A software reset is then issued to the GT processor i/f board and DIB through the DSPLINK interface to reset circuits and latches. The software reset operations for the GT processor i/f board and the DIB are described in [4] and [6] respectively.

A subroutine is called next to download the GT parameters for the simulation. The parameters are contained in an ASCII data file which is read and processed by the host/user interface program. The parameters are then transferred from the host to the DSP through the dual port memory. A copy of the ASCII data file for the GT parameters is included in Section C.4 of Appendix C.

The next step in the initialization process is to set up the numerically-controlled oscillator (NCO) to start the GT clock circuit on the GT processor i/f board. The clock circuit generates the hop clock and data clock signals for the GT simulator. The NCO setup is described in [4]. The HSC is also initialized in this part of the main assembly language program. The HSC initialization is carried out in the same manner as for the GT parameters. An ASCII file containing the HSC parameters is read by the host/user interface program which then passes the values to the DSP board. The DSP board then loads the values onto the HSC through the GT processor i/f board [4].

The final step in the initialization process is to enable the interrupts for the DSP board. An interrupt is generated on the rising edge of the hop clock in order that appropriate counters for the GT processor be updated and synchronization probes be generated if applicable. The interrupt service routine for the GT processor is described further in Section 3.3.1.4.

### 3.3.1.1.2 GT Simulator Modes of Operation

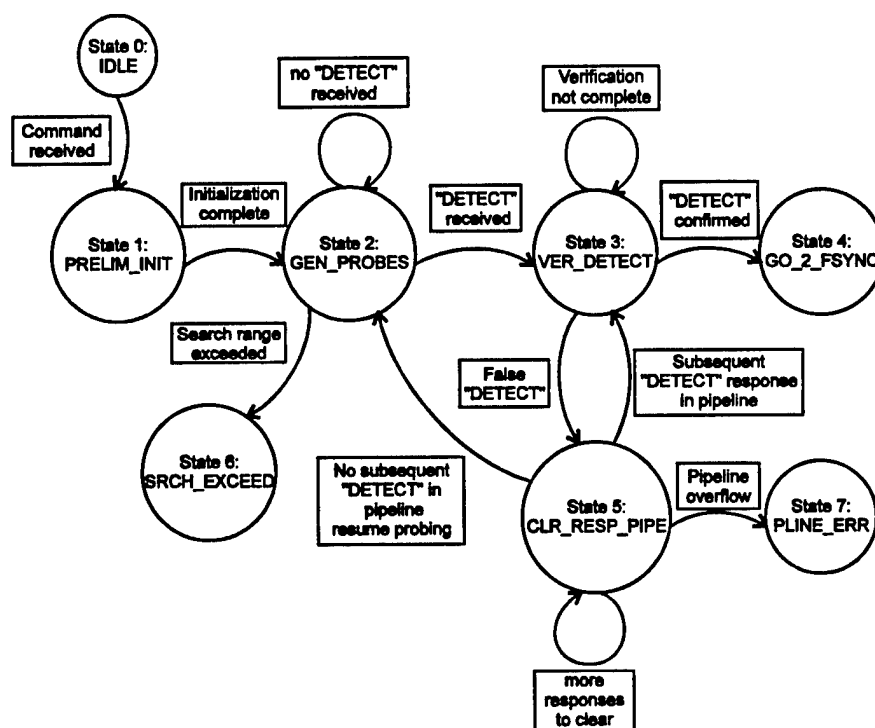
Once the DSP board and associated hardware are initialized, the main assembly program enters into a loop to wait for and respond to user commands transferred to it by the host/user interface program. The commands relate to different modes of operation for the GT simulator as well as to terminating the simulation. When a command is received, the DSP processes it and determines the appropriate subroutine to execute. If a command is received to terminate the simulation, the DSP disables the interrupts and remains idle. The modes of operation are listed in Table 3.2. The first five modes are discussed further in Section 3.3.2. A description of the algorithms for the coarse and fine synchronization modes is provided in the next sections.

Mode	Operation	Description
1	Transmitting a CW signal	GT simulator transmits one of five predefined continuous wave (CW) frequencies located in the frequency band of operation.
2	Run Mode	GT simulator switches the HSC to RUN mode [3] to randomly hop over the system bandwidth.
3	Sweep Mode	GT simulator sweeps a CW signal across the system bandwidth.
4	Frame Zero (FR0) Enable Mode	GT simulator monitors the downlink synchronization reference serial link for the downlink synchronization reference pulse and adjusts the GT hop clock counter if required.
5	FR0 Disable Mode	GT simulator ceases monitoring the downlink synchronization reference serial link.
6	Coarse Synchronization Mode	GT simulator performs coarse synchronization.
7	Fine Synchronizaion Mode	GT simulator performs fine synchronization.

**Table 3.2 GT simulator modes of operation**

### 3.3.1.2 Coarse Synchronization Assembly Routine

During coarse synchronization, the GT processor is responsible for several tasks. The tasks include generating coarse synchronization probes at appropriate times, processing synchronization responses transmitted from the payload simulator, and responding appropriately to these synchronization responses. When coarse synchronization is achieved, it is considered that the GT clock is within a hop from the payload system clock. The state diagram in Fig. 3.4 describes the general flow of events for the coarse synchronization process. The concepts or general procedure for the states shown in Fig. 3.4 are described in more detail in the following paragraphs. A listing of the assembly language program for the coarse synchronization routine is included in Section B.4, Appendix B.



**Fig. 3.4 Coarse synchronization procedure state diagram**

#### 3.3.1.2.1 Starting the Coarse Synchronization Procedure

In this implementation of the GT simulator, a command is issued by the user to start the coarse synchronization procedure. The command is issued from the host/user interface program which is described in Section 3.3.2. When the coarse synchronization command is received, the GT processor initializes parameters related to the transmission of synchronization probes and the processing of synchronization responses from the payload simulator. First, the GT processor must calculate the start time for the HSC. As described in [3], the delayed start time is necessary

in order to give the HSC ample time for the propagation of the random number generator and to precompute hop frequencies for the start frame. The start time is calculated based on a frame boundary, i.e. the start time is given as hop 0 of a frame "X". From [3], the earliest start time is calculated to be the third frame after the current frame. Where a terminal is not assigned all the coarse synchronization probe frames, the GT processor must wait until the first assigned frame following the delayed start time before transmitting the probes for the particular terminal. The start time is transferred to the HSC by way of the GT processor i/f board which is described in Section 3.2.2 and documented in [4].

When the start time is transferred to the HSC and the necessary parameters have been initialized for the coarse synchronization algorithm, the GT processor commands the HSC to switch to uplink synchronization mode and waits until the start time precalculation is completed before proceeding to state 2 to begin generating coarse synchronization probes.

### 3.3.1.2.2 Coarse Synchronization Probe Generation

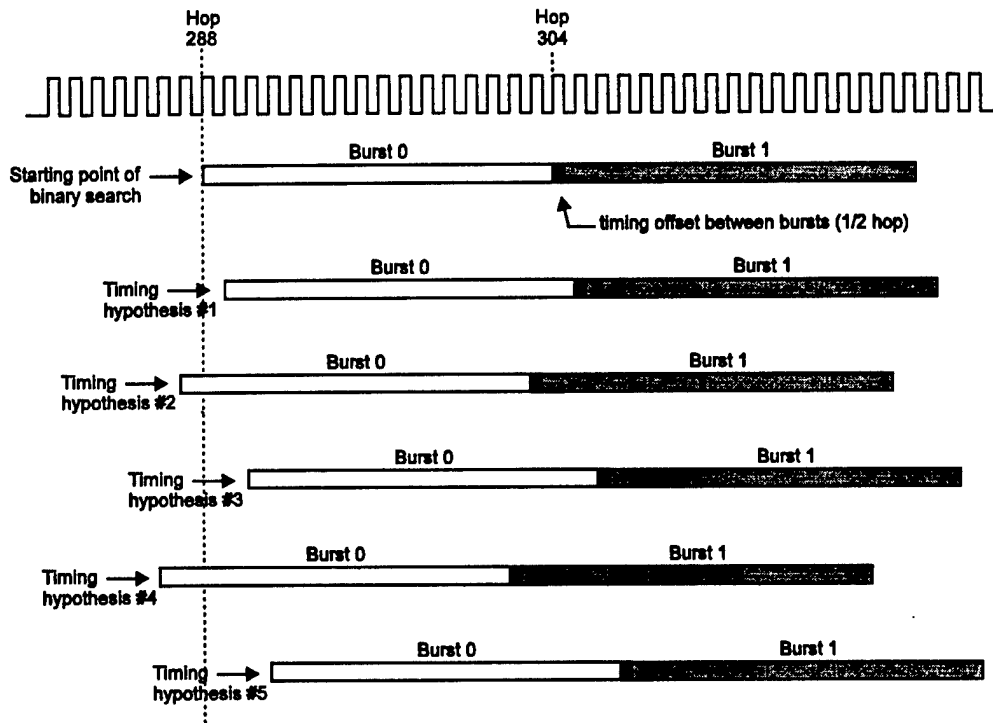
The GT processor's primary task in state 2 is to generate coarse synchronization probes at the appropriate time. In this implementation, only one terminal is realized to facilitate the demonstration of the coarse synchronization algorithm. In addition, the terminal allocation for transmitting coarse synchronization probes is selected to be on frames which are a multiple of four.

The process for achieving coarse synchronization consists of transmitting probes at different timing hypotheses. As described in Section 2.2.1, coarse synchronization probes are transmitted in the form of two bursts of sixteen probes each in the allocated channel and at the specific FSK tone bin. The coarse synchronization probes for the particular terminal implemented are to be transmitted on channel 1 and FSK tone bin 3. The two bursts of synchronization probes are transmitted with a timing offset between them. In this implementation, the second burst of synchronization probes is transmitted 1/2 hop later than the first burst of probes. If the received probes are detected by the payload processor, a "detect" response is formulated and relayed back to the GT simulator through the synchronization response return link. When the payload simulator is able to consistently detect the coarse synchronization probes for a timing hypothesis, it is considered that the timing of the GT clock is within a hop of the payload (system) clock.

The strategy selected to test the different timing hypotheses until the synchronization probes are detected involves starting at the most probable timing hypothesis and moving outward in increments of a hop at a time. The GT simulator uses the information obtained from the downlink synchronization pulse to derive the most probable timing hypothesis which does not account for any propagation delay. According to [2], the point at which synchronization probes are transmitted occurs at hop 288 of the time-frequency plan shown in Fig.2.3. As a result, the default starting point of the search for the correct timing hypothesis is selected to be at hop  $N=288$  of a terminal's assigned frame and corresponds to the case where there is perfect alignment of the GT clock with the payload clock.



Using the default starting point, the first series of coarse synchronization probes are transmitted starting on hop 288 of the terminal's assigned frame. On the next iteration, the synchronization probes are delayed by a hop and are transmitted starting at hop  $(N+1)=289$  of the terminal's next assigned frame. The following series of synchronization probes are transmitted a hop earlier than the starting point, at hop  $(N-1)=287$ . With each iteration of the search, the timing hypotheses move further away from the starting point. The "outward moving" search scheme is illustrated in Fig. 3.5.



**Fig. 3.5 Search scheme for coarse synchronization timing hypotheses**

The timing hypotheses are stored in an array indexed by the frame number during which the synchronization probes were transmitted. This allows a hypothesis to be retrieved should it result in the synchronization probes being detected by the payload simulator. The actions following the detection of synchronization probes are discussed below.

A search range is included as a parameter for the search scheme and is used to limit how many timing hypotheses are tested in the search. The search range is given by the absolute value of the maximum offset from the starting point for which coarse synchronization probes will be transmitted. The search range is in units of hops and is user-configurable as a parameter which is downloaded from an ASCII data file during run time. The ASCII data file is discussed in Section 3.3.3. The default search range is set at 32 hops which results in the probes being transmitted with timing hypotheses of up to  $\pm 32$  hops or  $\pm 1$  subframe. If the coarse synchronization probes have not been detected when the entire search range has been exhausted, the search range is

exceeded and the GT processor proceeds to state 6 where an error is signalled to the user. State 6 is described in Section 3.3.1.2.6.

In addition to generating the coarse synchronization probes, the GT processor processes the synchronization responses which are returned by the payload simulator. As described in Section 2.2.3, the synchronization responses for each frame are transmitted to the GT simulator via the synchronization response return serial link. The synchronization response contains both coarse and fine synchronization estimates. During coarse synchronization, the GT processor processes only the coarse synchronization estimate corresponding to the terminal's allocated probe frame. If no "detect" received in the appropriate coarse synchronization estimate, the GT processor does nothing further and continues with its task of generating coarse synchronization probes. However, if a "detect" is received in either of the bursts, the GT processor suspends the generation of synchronization probes for different timing hypotheses, and proceeds to state 3 to verify the "detect" received. The last timing hypothesis tested is saved before going to state 3 to facilitate resumption of the probe generation if the "detect" received proves to be an invalid one.

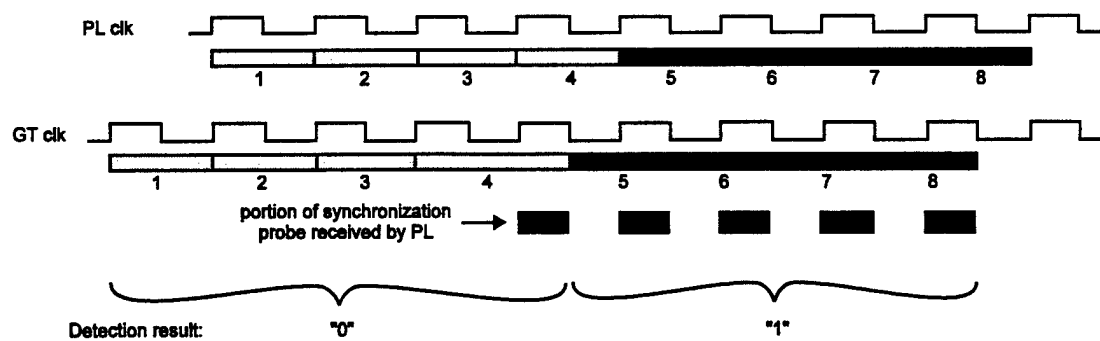
#### 3.3.1.2.3 Verification of "Detect" Responses Received

In order to be certain that a "detect" response is valid, the GT processor retransmits synchronization probes using the timing hypothesis which prompted the "detect". In this implementation, the GT processor uses the reference frame number included in the synchronization response to retrieve the appropriate hypothesis to be verified. The GT processor again waits until the terminal's allocated probe frame to retransmit the synchronization probes. The GT processor examines the subsequent coarse synchronization estimates to confirm the detection of synchronization probes by the payload simulator. Therefore, just as in state 2, the GT processor also processes synchronization responses from the payload simulator. For this implementation, the user can define the number of iterations for synchronization probe retransmission to validate the "detect" response. As well, the number of subsequent "detects" which are to be received to confirm detection is user-configurable. The two configurable values are included in the same ASCII data file that contains the search range parameter mentioned in Section 3.3.1.2.2.

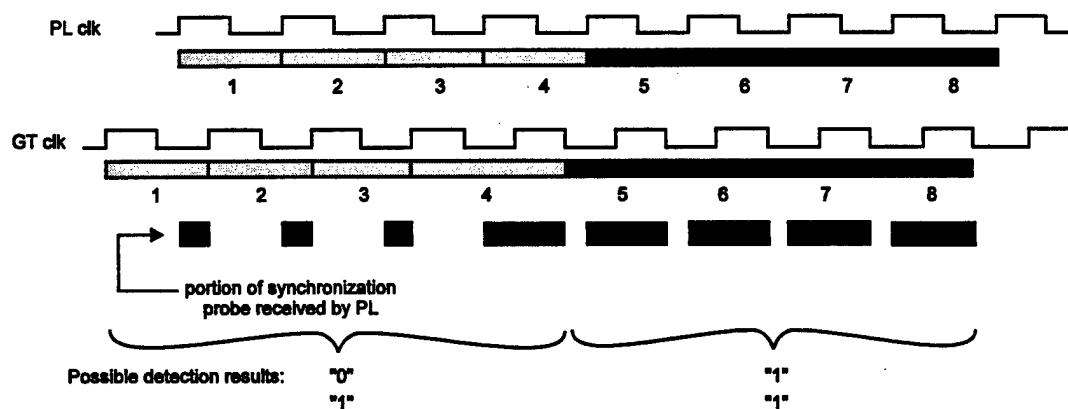
As shown in Fig. 2.4, there is a processing delay from the time the synchronization probes are transmitted to the time the corresponding synchronization response is received by the GT. As a result, when the GT processor first enters into state 3 to verify a "detect" response, the GT processor may continue to receive synchronization responses for probes sent prior to changing states. These synchronization responses are stored temporarily so that they may be processed later should a "false detect" result from the retransmitted synchronization probes during verification. A "false detect" occurs if the minimum number of "detects" required to confirm the detection for the retransmitted probes is not received for the number of iterations selected. If a "false detect" is concluded during the verification process, the GT processor enters into state 5 to process the temporarily stored synchronization responses. State 5 is described further in Section 3.3.1.2.5. In this implementation of the GT processor, only synchronization responses for the single terminal considered are stored in the synchronization responses buffer. As an added

precaution, the GT processor checks to see if there's any more room in the synchronization response buffer before storing the synchronization response. The overflow check serves as a debug feature to ensure the buffer is being cleared properly. If there is an overflow of the synchronization response buffer, then the GT processor goes to state 7 where a signal is sent to the user indicating the overflow error. The actions taken in state 7 are described further in Section 3.3.1.2.7.

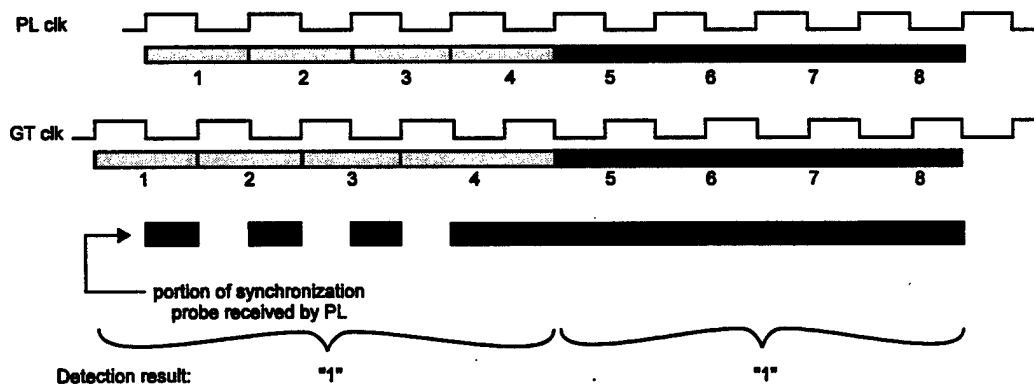
If the verification is successful and the original "detect" is confirmed, the GT processor examines "detect" responses for the retransmitted probes to determine whether the GT clock should be delayed by  $1/2$  a hop. This additional step is required because as mentioned in Section 3.3.1.2.2, the two bursts of synchronization probes are transmitted with a timing offset of  $1/2$  hop between them. In essence, two timing hypotheses are transmitted in one probe frame during coarse synchronization. As a result, the synchronization response from the payload simulator consists of two estimates for each coarse synchronization channel as described in Section 2.2.3. Fig. 3.6 illustrates the possible scenarios of synchronization responses. It is noted that four hops per burst are used to simplify the illustration. As the second burst is delayed by  $1/2$  a hop, if the responses for the retransmitted probes indicates a majority of "detects" in the second burst, the GT clock is delayed by  $1/2$  hop. Once the adjustment is made the GT processor advances to State 4 which is described further in the following section.



( i ) GT clock 1 hop earlier than payload clock

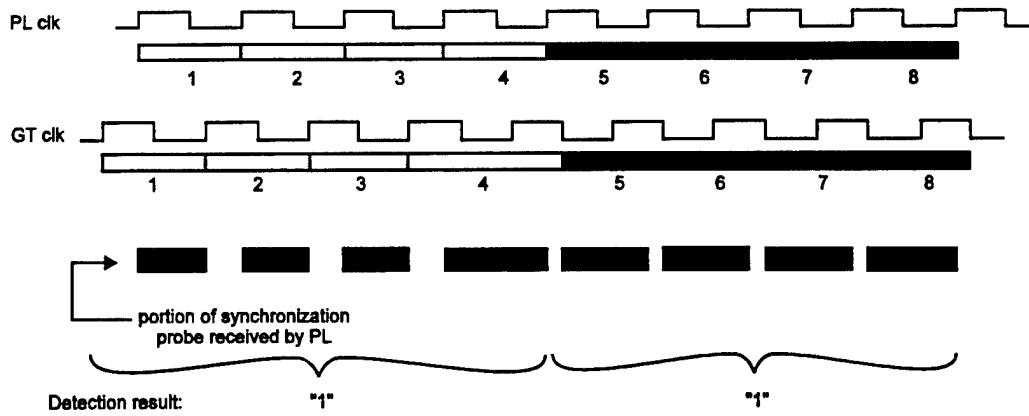


( ii ) GT clock between 1/2 and 1 hop earlier than payload clock

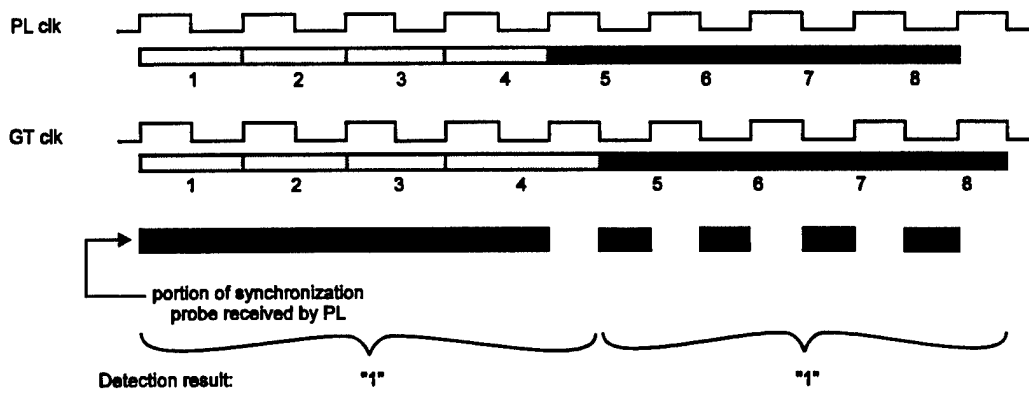


( iii ) GT clock 1/2 hop earlier than payload clock

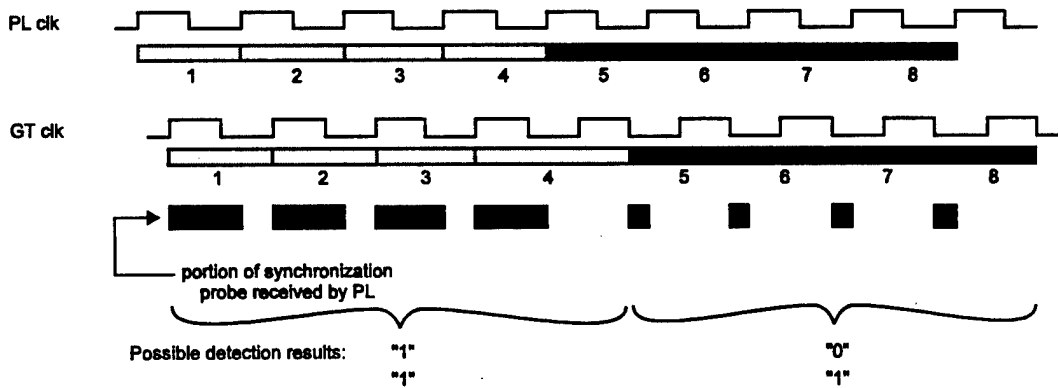
**Fig. 3.6 Different scenarios for GT clock alignment with payload clock**



( iv ) GT clock less than 1/2 hop earlier than payload clock

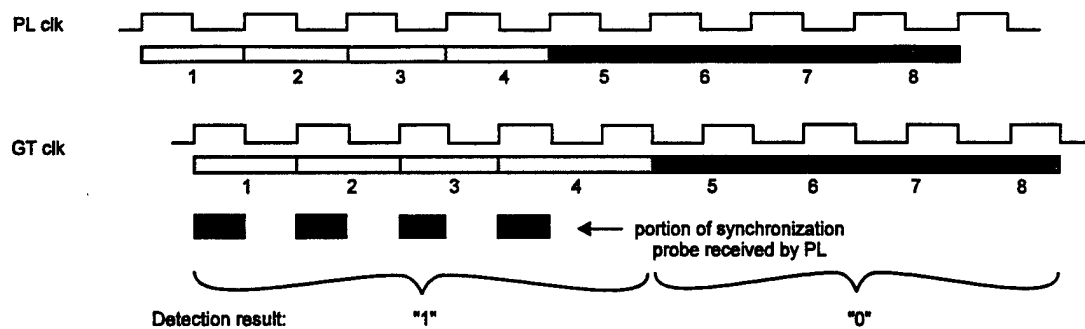


( v ) GT clock aligned with payload clock

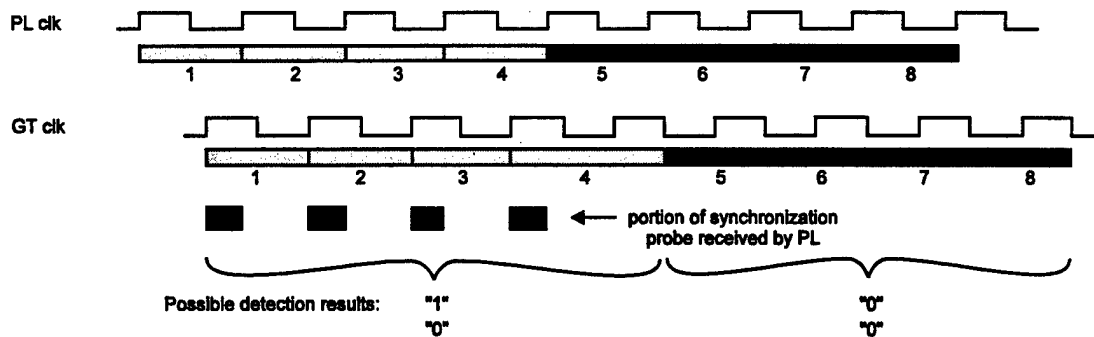


( vi ) GT clock less than 1/2 hop later than payload clock

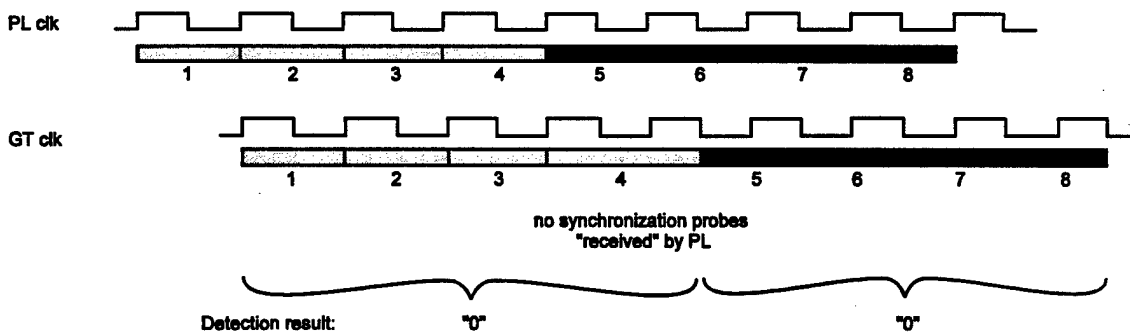
**Fig. 3.6 (cont'd) Different scenarios for GT clock alignment with payload clock**



( vii ) GT clock 1/2 hop later than payload clock



( viii ) GT clock between 1/2 and 1 hop later than payload clock



( ix ) GT clock 1 hop later than payload clock

**Fig. 3.6 (cont'd) Different scenarios for GT clock alignment with payload clock**

#### 3.3.1.2.4 Preparing for Fine Synchronization

In State 4, the GT processor uses the confirmed hypothesis to adjust the GT clock before proceeding to the fine synchronization algorithm. In this implementation, a signal is sent back to the user to indicate that coarse synchronization has been achieved. The user is then given the option to continue with fine synchronization or to halt the program. The user options are described in more detail in Section 3.3.2.

#### 3.3.1.2.5 Clearing the Synchronization Response Buffer

A synchronization response buffer is used to store synchronization responses which arrive after the GT processor changes from state 2 (generating synchronization probes) to state 3 (verification of "detect"). The responses correspond to the synchronization probes transmitted after those which resulted in a "detect" response from the payload simulator. If the original "detect" response is shown to be invalid in state 3, the GT processor processes the stored responses to see if a "detect" was received for the subsequent synchronization probes. The buffer thus saves the GT processor from having to retransmit synchronization probes for the hypotheses tested. If a "detect" is received in a subsequent response, then the GT processor switches back to state 3 to verify whether this "detect" is valid or not. Otherwise, if all the stored responses are processed and no "detect" occurs, then the GT processor resumes generation of synchronization probes (state 2).

#### 3.3.1.2.6 Search Range for Coarse Synchronization Routine Exceeded

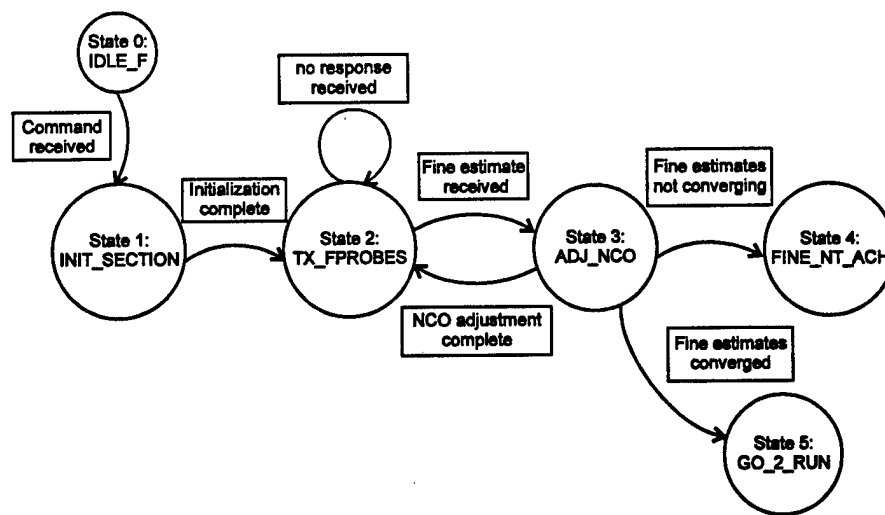
When synchronization probes for all the timing hypotheses have been transmitted for the search range and no subsequent "detect" is received for any of the probes, a flag is set to alert the user. The user is then given a choice to repeat the downlink synchronization procedure or to halt the program. The user options are described in more detail in Section 3.3.2.

#### 3.3.1.2.7 Synchronization Response Buffer Overflow

If an overflow occurs in the synchronization response buffer, then a flag is set to alert the user to the error. The host/user interface program for the GT simulator is subsequently aborted. The host/user interface program is described further in Section 3.3.2.

#### 3.3.1.3 Fine Synchronization Assembly Routine

Once coarse synchronization is achieved, the GT performs fine synchronization in order to refine the GT clock alignment with the payload clock. For fine synchronization, the GT processor performs tasks similar to those during coarse synchronization. The GT processor generates fine synchronization probes to be transmitted at allocated times and processes synchronization responses transmitted by the payload simulator relating to the fine synchronization probes. During fine synchronization, the GT processor is also responsible for adjusting the GT clock according to the fine synchronization estimate received from the payload simulator. For this implementation, fine synchronization is considered achieved when the GT clock is within 10% of a hop from the payload clock. The choice of 10% was considered reasonable to account for frequency drift with minimal degradation to FSK modulation performance. The fine synchronization procedure is illustrated in Fig. 3.7. The general concepts of the states in Fig. 3.7 are described further in the following subsections. A listing of the assembly program for the fine synchronization routine is included in Section B.5, Appendix B.



**Fig. 3.7 Fine synchronization procedure state diagram**

#### 3.3.1.3.1 Beginning Fine Synchronization

Upon receiving a command to perform fine synchronization, the fine synchronization routine begins by initializing variables associated with the transmission of fine synchronization probes. The DSP also sends a command to the HSC to go to RUN mode after which the HSC switches the frequency synthesizer according to a pseudorandom sequence [3]. The DSP then proceeds to state 2 to begin transmitting fine synchronization probes.

#### 3.3.1.3.2 Generation of Fine Synchronization Probes

As in the case of coarse synchronization, only one terminal is realized to facilitate the demonstration of the fine synchronization algorithm. In addition, the terminal allocation for transmitting fine synchronization probes is again selected to be on frames which are a multiple of four.

The fine synchronization process involves transmitting fine synchronization probes during a terminal's allocated frame and time slot. As described in Section 2.2.2, fine synchronization probes are transmitted in a single burst of 32 probes. Another difference between the two synchronization processes is in the way the probes are transmitted. For coarse synchronization, synchronization probe frequencies are precomputed and the GT processor sends commands to the HSC to transmit the probes. In fine synchronization, the frequency for the fine synchronization probe is transferred to the HSC on a hop basis. The fine synchronization probe frequency is defined by a channel and FSK tone bin. The channel and FSK tone bin information are passed to the HSC through the GT processor i/f board. The HSC then uses the information to



generate the appropriate transmit hop signal on the frequency synthesizer. In this implementation of the GT simulator, the fine synchronization probes are transmitted on channel 2, FSK tone bin 3. The GT processor's task is to determine when it is time to transmit the fine synchronization probes. The transmission of the fine synchronization probes is carried out by the interrupt service routine which is described in Section 3.3.1.4. When it is time to transmit fine synchronization probes, the GT processor sets a flag which is monitored by the interrupt service routine.

In addition to determining when fine synchronization probes are to be transmitted, the GT processor processes the synchronization responses which are returned by the payload simulator. Again, the synchronization responses for each frame are transmitted to the GT simulator on the synchronization response return serial link. The fine synchronization responses correspond to the payload simulator's estimate of how early or late the received probes are relative to the system clock. In this implementation, the host/user interface program of the GT simulator initially processes the fine synchronization responses and computes an average of ten responses received for the user. The host/user interface program also calculates a corresponding phase adjustment required for the NCO to refine the GT clock alignment. The average estimate of the responses and the phase change are then transferred to the GT processor. When the GT processor has read and stored the two values, the GT processor proceeds to state 3 to adjust the GT clock.

#### 3.3.1.3.3 Refining the Alignment of the Ground Terminal Clock

Prior to adjusting the NCO phase to realign the GT clock, the GT processor checks whether the estimates received for the user are converging. The point of convergence is chosen to correspond to the GT clock being within 10% of a hop from the payload clock. In addition, the GT processor checks to see how many times the GT clock has been adjusted (or how many bursts of fine synchronization probes have been sent) to avoid the situation of endlessly trying to perform fine synchronization.

In this implementation, the adjustment of the GT clock is carried out over a frame (320 hops) to minimize frequency discontinuities while making the adjustment in a timely manner. Based on the estimate received from the synchronization response, a new GT clock frequency is calculated to align the GT clock over a period of 320 hops, after which the original GT clock frequency is reinstated. The calculations for the new frequency which take into consideration this gradual approach is described in Section 3.3.2. If the synchronization response indicates that the probes received were early, the GT clock frequency is lowered to, in effect, delay the clock. Conversely, if the estimate shows that fine synchronization probes were late, then the GT clock frequency is increased to advance the clock. The clock frequency adjustment is implemented by adjusting the phase increment of the NCO which produces the clock signals for the GT simulator. In the assembly language program, a subroutine is called to transfer the new phase increment to the NCO via the GT processor i/f board. Once the new phase adjustment is loaded, the GT processor waits for 320 hops after which the original phase increment for the NCO is reloaded to produce the initial GT clock frequency.

#### 3.3.1.3.4 Nonconvergence of Fine Synchronization Estimates

The arrival into this state implies that either the synchronization estimates being received from the payload simulator are not converging, or it has taken too long for the synchronization estimates to converge. At this point, the assembly language routine sets a flag to indicate to the host program that this event has occurred. The fine synchronization assembly language routine then returns to the main assembly language program to wait for the next command from the user.

#### 3.3.1.3.5 Achieving Fine Synchronization

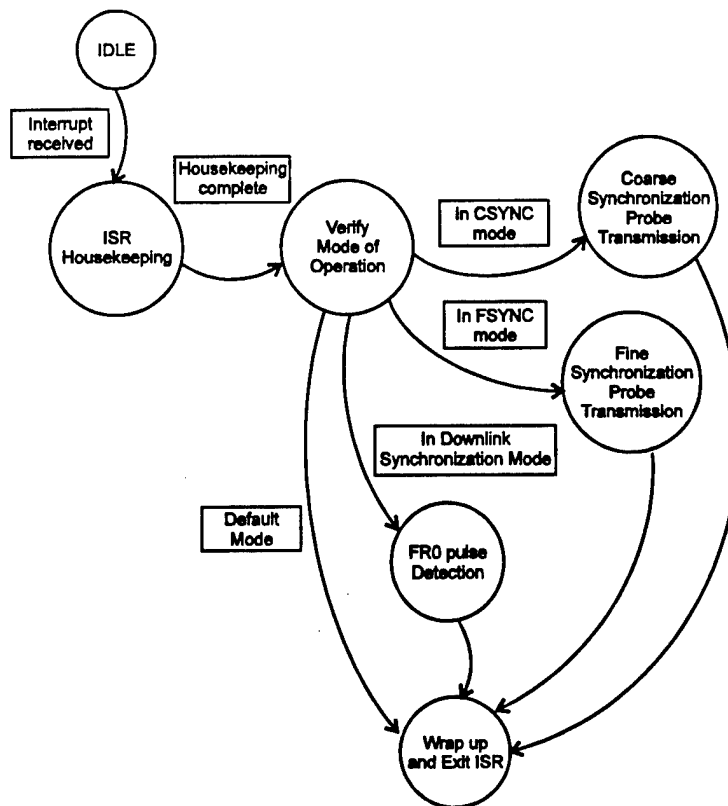
When fine synchronization is achieved, the GT processor sets a flag to the host program. At this point, the GT processor is ready to transmit user data. However, the development of the GT simulator was terminated here as the demonstration of synchronization concepts was the primary goal of this project.

#### 3.3.1.4 Interrupt Service Routine

In the GT simulator implementation, an interrupt is generated on the rising edge of every hop clock pulse. When an interrupt is received by the DSP, the DSP stops its current task and invokes the interrupt service routine (ISR). The interrupt service routine carries out some routine updating of counters and performs tasks related to synchronization if applicable. Once the interrupt service routine has been executed, the DSP continues the task it was performing prior to receiving the interrupt. The interrupt service routine is written in assembly language and can be found in Section B.6, Appendix B. The tasks for the interrupt service are shown in Fig. 3.8 and are described in more detail in the following paragraphs.

##### 3.3.1.4.1 ISR Housekeeping

The first step of the interrupt service routine consists of clearing the interrupt signal sent to the DSP by the GT processor i/f board. The interrupt signal is automatically cleared on the GT processor i/f board by reading the interrupt port of the GT processor i/f board [4]. The next step for the interrupt service routine is to update the hop and frame counters of the GT processor. Finally, the interrupt service routine checks to see if the GT processor is in either downlink synchronization, uplink coarse synchronization, or uplink fine synchronization modes. If the GT processor is not in one of these modes, the interrupt service routine is then complete and the DSP can resume its normal operations. Otherwise, the interrupt service routine performs one of the tasks described below before the DSP resumes its normal operations.



**Fig. 3.8 Interrupt Service Routine Flow Diagram**

#### 3.3.1.4.2 ISR Tasks for Coarse Synchronization Mode

For coarse synchronization, the interrupt service is responsible for transmitting the coarse synchronization probes at the appropriate time. In order to determine the appropriate time for transmitting the coarse synchronization probes, the interrupt service routine goes through a series of checks. As described in [3], when the HSC is changed to CSYNC (coarse synchronization) mode, the HSC first requires a certain amount of time to precompute the hop frequencies for the coarse synchronization probes. Thus, the interrupt service routine first verifies that the precomputation time has elapsed. After the precomputation time has elapsed, the interrupt service routine must check to see if the current hop is hop 0. During coarse synchronization, the HSC must be given an FCALC (frequency calculate) command on hop 0 of every frame in order to calculate the next synchronization probe frequencies. The next step is to verify whether the current frame is one that is assigned to the user. If it is an assigned frame, the interrupt service routine checks to see if it is time to transmit coarse synchronization probes based on a particular timing hypothesis. Once it is the appropriate time, the interrupt service routine must differentiate between transmitting one of the two bursts of probes with a timing offset of 1/2 hop between them as described in Section 3.3.1.2.2. The transmission of the coarse synchronization probes is

accomplished by sending ULGO (uplink GO) commands to the HSC through the GT processor i/f board [3, 4].

#### 3.3.1.4.3 ISR Tasks for Fine Synchronization Mode

As described in Section 2.2.2, during fine synchronization, the GT processor transmits a series of bursts of 32 fine synchronization probes at specific times allocated to a particular terminal. As a result, the interrupt service routine must perform a similar series of checks as for coarse synchronization in order to determine when to transmit the synchronization probes. First, the interrupt service routine checks to see if the current frame is an allocated frame for fine synchronization. If so, the interrupt service routine checks for the appropriate time to transmit the fine synchronization probes. The fine synchronization probes are transmitted by writing the channel and FSK bin information to the FSK/FRAME port of the GT processor i/f board [3, 4]. If the current frame is not allocated for the transmission of fine synchronization probes, a TXOFF (transmit off) command is written to the FSK/FRAME port which causes the transmit signal to be attenuated, effectively turning the transmitter off.

#### 3.3.1.4.4 ISR Tasks for Downlink Synchronization Mode

As described in Section 2.1, downlink synchronization is performed in order to acquire the satellite downlink. In practice, downlink synchronization consists of the ground terminal detecting synchronization aids transmitted by the payload. The detection of the synchronization aids also provides the ground terminal with some preliminary information to begin uplink synchronization. A simulated downlink synchronization link was implemented whereby a reference pulse is transmitted on a serial link connecting the payload and GT simulators. The rising and falling edges of the reference pulse were chosen to correspond to the start of hop0/frame0 and hop0/frame 1 respectively. Access to the reference pulse is achieved by way of the status register on the DIB [6]. When the ground terminal processor is in downlink synchronization mode, the interrupt service routine reads the status register of the DIB and examines whether a rising or falling edge has occurred. If either edge is detected, the interrupt service routine verifies the hop and frame counters are set properly or resets the counters as required.

### 3.3.2 Host/User Interface Program

The user interface to the GT simulator is provided by the host/user interface program which is written in C. A listing of the host/user interface program is included in Section B.2, Appendix B. The host/user interface program initializes and downloads the DSP code to the DSP board (also referred to as the GT processor) and allows the user to select the mode of operation for the GT simulator. Furthermore, the host/user interface program monitors the status of the simulation by continuously checking a series of flags and responding as required. The general flow of the program steps is shown in Fig. 3.9. A description of each of these steps or tasks is provided in the following paragraphs.



**Fig. 3.9 Host/user interface flow diagram**

#### 3.3.2.1 DSP Board Initialization

The first step of the host/user interface program involves using interface library subroutines provided by Spectrum Signal Processing Inc. [11] to properly initialize the DSP board which operates as the GT processor. Once the appropriate DSP board is selected and initialized, the DSP code is downloaded into the DSP program memory. The dual port memory of the DSP board, which allows for data transfer between the DSP and host PC and vice versa, is then initialized. Finally, a reset is issued to the DSP board using the interface library functions to start the execution of the DSP program.

#### 3.3.2.2 Downloading Simulation Parameters

In order to allow flexibility in changing parameter values for a particular simulation, parameters are stored in ASCII files which are read by the host/user interface program and subsequently transferred to the DSP board. For this implementation of the GT simulator, there are three ASCII data files. The three data files include values for general GT parameters, for the HSC, and for CW frequency values. The data files are described further in Section 3.3.3 and are included in Appendix C. In the host/user interface program, each value is read and then formatted to be downloaded to the DSP. The downloading of the parameter values is also reflected in the DSP main assembly program which is discussed in Section 3.3.1.1.1.

### 3.3.2.3 Serial Communications Initialization

As described in Section 3.1, the experimental setup consists of the ground-based payload and GT simulators being located approximately 1.5 km apart. In order to facilitate integration and testing of the software, a remote operation capability was implemented for the GT simulator. A serial communications link was installed to connect the two simulators. Using the serial communications software developed at DREO [10], the GT simulator can be operated by the payload simulator. It is noted that the scope of the remote operation only encompasses the mode of operation for the GT simulator. The GT simulator still has to be powered on and the executable file has to be run locally before remote operation can occur. Similarly, if a catastrophic error occurs and a reboot of the GT simulator is required, the reboot must also be performed at the local terminal.

### 3.3.2.4 User Interface Menu

During the development of the GT simulator, several modes of operation were implemented. The modes of operation were introduced in Section 3.3.1.1.2. While some of these modes are not part of the uplink synchronization process, they are retained for debug purposes. A mode is selected by the user from a menu which is displayed on the local terminal. A replica of the user menu which is displayed on the local screen is shown in Fig. 3.10. Each of the options in the menu are described further in the following paragraphs.

```
*****
*                               *
*           CW test - 7-11 April 1997           *
*           GT Synch Processor Menu             *
*                               *
* Enter one of the following:                   *
*                               *
* 'L' : Go to lower edge of hop BW              *
* 'U' : Go to upper edge of hop BW              *
* 'M' : Go to middle of hop BW                  *
* 'Q' : Go to one quarter mark of hop BW        *
* 'T' : Go to three quarter mark of hop BW      *
* 'F' : Go to specific frequency                *
* 'R' : Go to RUN mode                          *
* 'S' : Slowly cycle through hop BW              *
* 'E' : Enable interrupt/FRO detection           *
* 'D' : Disable interrupt/FRO detection          *
* 'C' : Coarse Synchronization test             *
* 'W' : Fine Synchronization test               *
* 'X' : Exit program or stop slow hopping (option 'S') *
* TMS interrupts/FRO detection is currently disabled. *
*****
Enter selection:
```

**Fig. 3.10 User menu for the GT simulator**

The first six options on the user menu allow the user to select a CW frequency tone to be transmitted. The first five of six options ('L', 'U', 'M', 'Q', 'T') correspond to the lower edge, the upper edge, the middle, the quarter mark, and the three-quarter mark of the hopping bandwidth respectively. These values are stored in an ASCII data file which is read by the host/user program as described above. The sixth option ('F') corresponds to an arbitrary value within the hopping bandwidth, selectable by the user.

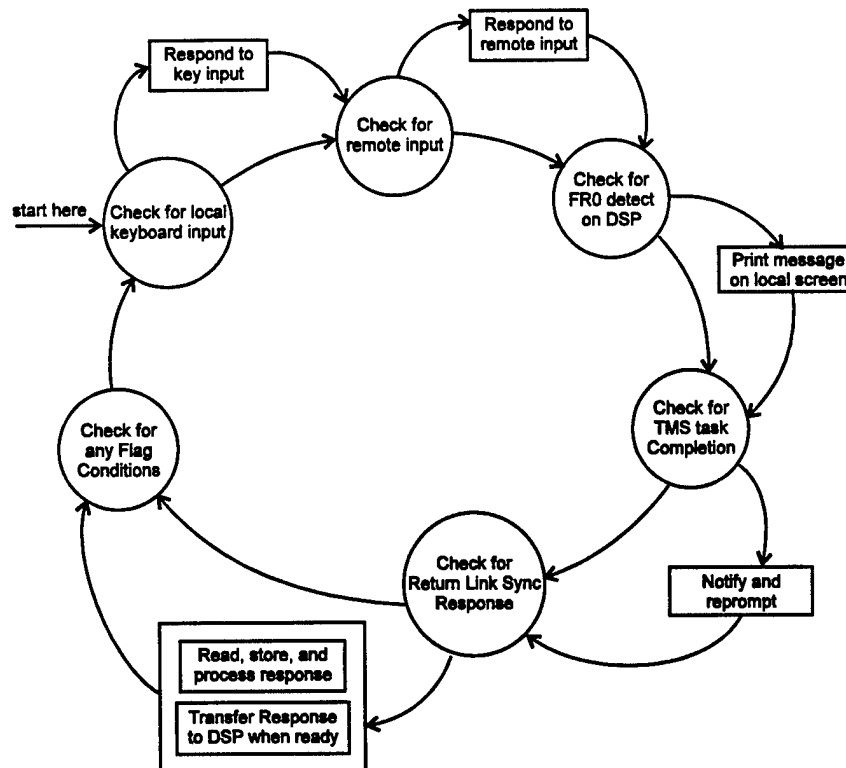
The RUN mode option ('R') causes the HSC to randomly hop over the hopping or system bandwidth. Alternatively, the 'S' option transmits a CW tone that is swept across the hopping bandwidth. The user can select the rate at which the tone is swept through the bandwidth by entering a dwell time (in seconds) and a hopping increment (between 1 and 16777215) for the prompts following the 'S' entry.

The 'E' option enables the detection of the downlink synchronization reference pulse (FR0 pulse). When this option is invoked, the DSP returns a flag every time the rising edge of the FR0 pulse is detected. The host/user interface program prints a message on the local screen indicating that the rising edge of the FR0 pulse was detected and prints the values of the hop and frame counters when the edge is detected. The values indicate whether an adjustment is required for the counters. When no adjustments are observed for the counters over a period of time, downlink synchronization can be considered to be achieved and thus, the detection of the FR0 pulse can be disabled. To disable this option, a 'D' may be entered at any time during this mode of operation.

The 'C' and 'W' allow the user to execute the coarse and fine synchronization algorithms for the GT simulator, respectively. The detailed description of the algorithms are included in Sections 3.3.1.2 and 3.3.1.3.

### 3.3.2.5 Host/User Interface Loop

Once the DSP and host initialization have taken place and the user menu is displayed, the host enters into a loop operation to respond to user commands and to flag conditions relating to the status of the simulation. Fig. 3.11 illustrates the loop. The order in which the loop is carried out is only a reflection of the order in which the GT processor functions were developed.



**Fig. 3.11 Host/user interface loop**

#### 3.3.2.5.1 Checking for User Input - Local Keyboard Input

If the host detects a keyboard entry, it will process the input to determine whether the input is valid and if so, take appropriate action. The valid entries are the options on the user menu described in Section 3.3.2.4. The subroutine for processing keyboard entries has been written to be case insensitive (i.e. the host program interprets upper and lower case entries to be the same). If a valid entry is received, the host transfers the appropriate command to the DSP board. In addition, if serial communications between the payload and GT simulators is activated, a keyboard entry at the local terminal will also cause a message to be sent to the remote terminal identifying the entry made. If an invalid entry is received, the host program prints an error message, prompts the user for another entry and displays the user menu again.

#### 3.3.2.5.2 Checking for User Input - Remote Input

The host program also checks for remote input if serial communications is enabled between the payload and GT simulators. Similar to any keyboard input, the host program processes the remote input to determine whether it is valid. The input options for remote input correspond to the options on the user menu. Remote input through the serial communications



link is formatted as text messages which must be read and decoded by the host program. If a valid entry is received, the host transfers the appropriate command to the DSP board and prints a message on the local terminal that a remote entry has been made.

#### 3.3.2.5.3 Checking for FR0 Pulse Edge Detection

The task of checking for the FR0 pulse edge detection was originally used in the development of the downlink synchronization mode for the GT simulator. The task has been retained for debug purposes. The task involves reading a location in dual port memory to see if the flag for an FR0 pulse edge detection is set. If the flag is set, the host program clears the flag and displays a message on the screen including the value of the hop and frame counters before the edge was detected. The detection of the FR0 pulse can be disabled at any time by entering 'D'.

#### 3.3.2.5.4 Checking for Completion of Task by DSP

In this step of the loop operation, the host program checks to see whether the DSP is ready to receive another user command by reading the C30DONE flag which is also stored in dual port memory. When the C30DONE flag is set, the host program prompts the user for another command.

#### 3.3.2.5.5 Checking for Return Link Synchronization Response

Synchronization responses are received by the GT simulator from the payload through a serial communications link and stored in a buffer on the host PC. When a synchronization response is received, the host program reads and decodes the response according to the format presented in Section 2.2.3. If the GT simulator is in coarse synchronization mode, the response is transferred to the DSP. However, in the case of fine synchronization, an average of ten fine synchronization responses is computed first. An average value of the fine synchronization estimates was chosen over a single estimate to provide further accuracy of the realignment required for the GT clock and possibly reduce the number of adjustments needed to align the GT clock with the payload clock. The average is then used to compute the phase change required on the NCO to align the GT clock. Both the average fine synchronization response and the corresponding phase change required are then transferred to the DSP.

#### 3.3.2.5.6 Checking Other Flag Conditions

The final step of the loop operation for the host program is to check a series of flags which indicate the status of the simulation. Again, while some flags may not explicitly be part of the uplink synchronization process, they are retained for debug purposes. Table 3.3 describes each of the flags which are monitored by the host program and any subsequent options should a

flag become set. Again, the order in which the flags are listed only reflect the order in which the functionality was added in the development of the GT simulator.

### 3.3.3 ASCII Data Files

To facilitate the reconfiguration of parameters for the GT processor and the HSC during the trials without the need to recompile code, three ASCII data files are used. These data files must be stored in the same directory as the executable file for the host/user interface C program. The first data file called "freq.dat" contains the frequency values at specific points within the hopping bandwidth. The frequency values are used to generate CW tones to verify the system integration of the GT and payload simulators as well as to verify the SATCOM link. Frequency values are included for both the COMSTRON and SCITEQ frequency synthesizers which are supported in the uplink synchronization trials. The second data file, "hscinit.dat", contains the parameters to be initialized on the HSC. Again, initial values are included for both the COMSTRON and SCITEQ synthesizers. The third data file called "GTparam.dat" allows the user to select values for GT processor variables. The user-definable variables enable changes to the coarse synchronization and fine synchronization procedures. These include the search limit for the synchronization probe hypotheses, the number of times a "detect" must be received to confirm coarse synchronization, the synchronization response buffer size, and the range of fine synchronization estimates within which fine synchronization is considered to be achieved. The ASCII files are included in Appendix C.

Flag Name	Description
RNG_XCDED	Indicates that the search range for the coarse synchronization algorithm has been exceeded with no coarse synchronization probes being detected. The search range is defined by the user in an ASCII data file which is read by the host program. This flag is used to indicate that the range of timing hypotheses to be tested has been exceeded. The occurrence of this flag being set may indicate the possibility of an error with the downlink synchronization acquisition. Thus, in the event this flag is set, the user is given the option to return to downlink synchronization mode.
CSYNC_OK	Indicates that coarse synchronization has been achieved. At this point, the detection of coarse synchronization probes for a particular timing hypothesis has been confirmed. The timing hypothesis is used to adjust the hop and frame counters (i.e. GT clock). If this flag becomes set, the user is given the option to go to fine synchronization mode.
PLINE_FLAG	Indicates that the buffer used by the GT processor to store coarse synchronization responses is full. The buffer is used when the GT processor is verifying a timing hypothesis which resulted in a "detect" response from the payload. The buffer holds only responses for timing hypotheses which were transmitted after the timing hypothesis which resulted in the "detect" response but before the verification procedure commenced. The verification of timing hypotheses is described in Section 3.3.1.2.3. This flag is used for debug purposes.
TOO_MANY_HYPS	Indicates that the buffer used by the GT processor to store the reference frame number and timing hypotheses is full. This condition occurs when the GT processor tries to overwrite another entry. This flag is used for debug purposes.
FRM_NOT_FOUND	When a coarse synchronization response is received from the payload, it is only referenced by the frame number to which the response corresponds. Thus, the GT processor must retrieve the timing hypothesis used in that particular frame to verify the hypothesis. The GT processor retrieves the timing hypothesis from the hypothesis log. If the frame number is not found and the timing hypothesis cannot be retrieved, the FRM_NOT_FOUND flag is set. The host program is subsequently aborted. This flag is used for debug purposes.
FSTART_AVAIL	Indicates that the frame number of the first valid fine synchronization response has been transferred by the GT processor. The frame number is stored by the host program and used to start processing fine synchronization responses.
FSYNC_OK	Indicates that fine synchronization has been achieved. A message is printed on the screen that the GT is ready to transmit data. Due to time constraints, the data transmission function has not been implemented on the GT simulator.
NO_FSYNC	Indicates that fine synchronization could not be achieved as the fine synchronization responses did not converge. If this condition occurs, the user is given the option to go back to coarse synchronization.
UFLO_CDTN	Indicates that data was not transferred to the GT processor i/f board in time for the next hop during fine synchronization, resulting in an underflow condition. A message is printed on the local terminal screen and the host program is subsequently aborted.

**Table 3.3 Description of flags for GT simulator**

## 4.0 Summary

This report describes the development of a GT simulator for an in-house activity examining synchronization aspects of EHF SATCOM at DREO and CRC. The GT simulator consists of a GT processor, a number of custom interface boards, frequency synthesizer, RF equipment, and a data source. The GT processor was implemented on a TMS320C30 DSP board from Spectrum Signal Processing Inc., and is contained in a host PC. A GT processor i/f board was designed and fabricated to generate the necessary clock signals for the GT processor and to provide the interface between the GT processor and an HSC. The HSC, in turn, controlled a frequency synthesizer used to produce the transmit signal. A multipurpose data interface board was also designed and fabricated to provide the interface between the GT processor and a data source, and to provide the interface to the downlink synchronization reference link.

The modes of operation for the GT simulator include: transmitting CW tones at specific points in the hopping bandwidth; transmitting an arbitrary CW tone within the hopping bandwidth; sweeping a CW tone across the hopping bandwidth; performing downlink synchronization using a simulated downlink; performing uplink coarse synchronization; and performing uplink fine synchronization. The first three functions were implemented during the development of the simulator and are retained for debug purposes.

The GT simulator was developed for use in uplink synchronization trials over the UK Skynet 4A satellite. The synchronization procedure begins with downlink synchronization. In this implementation, a simulated downlink is setup between the payload and GT simulators via an RS232 serial communications link. A downlink synchronization reference pulse is continuously transmitted by the payload simulator to the GT simulator. The edges of the reference pulse correspond to specific instances in the pseudorandom hopping sequence of the payload simulator. The GT simulator detects and uses this reference pulse to begin uplink synchronization.

There are two stages in performing uplink synchronization: coarse and fine synchronization. Both involve the transmission of synchronization probes to the payload simulator at allocated times. Synchronization responses to the synchronization probes are formulated by the payload and returned to the GT simulator. In coarse synchronization, a binary "detect/no detect" response is returned. In fine synchronization, a time estimate of how early or late the received probes are relative to the system clock is provided on the synchronization response return link. The synchronization response return link is implemented on a serial communications link capable of data transfer at 9.6 kb/s. These responses are transmitted to the GT simulator once every frame.

For coarse synchronization, two consecutive bursts of sixteen synchronization probes are transmitted at different timing hypotheses. In this implementation of the GT simulator, an "outward moving" search scheme is used in coarse synchronization to test different timing hypotheses beginning with the most probable hypothesis. In addition, a verification process is

carried out when a “detect” response is first received by the GT simulator for a particular timing hypothesis. Coarse synchronization is considered achieved when the GT clock is aligned to within a hop of the payload clock.

In fine synchronization, a single burst of thirty-two synchronization probes is transmitted. In order to reduce the number of times the GT clock is adjusted and to minimize any estimate errors due to noise, an average of ten fine synchronization responses is used to determine the fine adjustment required by the GT clock during fine synchronization. Fine synchronization is considered achieved when the GT clock is aligned to within 10% of a hop of the payload clock.

The GT simulator has been developed so that it can be remotely operated by the payload simulator once the GT simulator is powered on and the executable file is run. The remote operation capability is included to facilitate the operation of the trials since the GT and payload simulators are physically located 1.5 km apart.

Data files containing parameters for the simulation are used to allow changing of their values without recompilation. The parameters are downloaded by the host PC to the DSP at the beginning of the simulation.

A user’s guide is included in Appendix A for installation and operating procedures of the GT simulator. Appendix B contains a listing of the software programs used by the GT simulator. Appendix C contains a listing of the parameter data files used by the GT simulator.

## References

- [1] Addison, R.D., Seed, W.R., "*Implementation of an EHF Frequency-Hopping Simulator*", DREO Report 1279, Ottawa, Canada, December 1995.
- [2] Lambert, J.D., "*DREO/CRC Joint Data Link Standard for Low Data Rate Service to EHF Ground Terminal and Payload Simulators*", DREO Report 1069, Ottawa, Canada, February 1991.
- [3] Addison, R.D., "*Modified Hopping Synthesizer Controller*", DREO Report 1304, Ottawa, Canada, December 1996.
- [4] Tom, C., "*Ground Terminal Processor Interface Board for Skynet Uplink Synchronization Trials*", DREO Report 1321, Ottawa, Canada, November 1997.
- [5] Tom, C., Meng, Z., "*Multichannel M-ary Frequency-shift-keying Block Demodulator Implementation*", DREO Report 1307, Ottawa, Canada, December 1996.
- [6] Simoneau, Y., Tom, C., "*Multipurpose Data Interface Board*", DREO Report 1332, Ottawa, Canada, October 1998.
- [7] *Series FS2000 Frequency Synthesizer Operation and Maintenance Manual*, Comstron Corporation, 1987.
- [8] *VDS-3000-977 Frequency Synthesizer Operating Instructions*, Sciteq Electronics Inc., Rev.A, January 14, 1983.
- [9] *Model 1645A Data Error Analyzer Operating and Service Manual*, Hewlett Packard Company, November 1983.
- [10] Addison, R.D., "*Real-time Interprocessor Serial Communications Software for Skynet EHF Trials*", DREO Report 1227, Ottawa, Canada, July 1994.
- [11] *TMS320C30 System Board User's Manual*, Spectrum Signal Processing Inc., Issue 1.01, August 1990.

## **Appendix A: Ground Terminal Simulator Installation Guide**

### **A1 Installation**

#### **A1.1 Hardware installation**

The GT simulator (excluding the RF components) consists of the following:

- IBM-AT compatible PC
- Spectrum Signal Processing Inc., TMS320C30 DSP Board (GT processor)
- GT processor i/f board
- Hopping synthesizer controller
- Frequency synthesizer (COMSTRON or SCITEQ)
- Multipurpose DIB
- Data source (HP1645 Bit Error Rate Analyser)
- DSPLINK backplane cage

The DSP board or GT processor is installed in an ISA expansion slot of the PC. The configuration and jumper settings of the DSP board are described in [11]. For the uplink synchronization experiments, the default settings for the DSP board are used. The GT processor i/f board and the multipurpose DIB are each inserted into a slot of the 96-pin DSPLINK backplane. The DSPLINK backplane was described in Section 3.2.7.1 of the main document. Details of the configuration of the GT processor i/f board and DIB are provided in [4] and [6] respectively. Communications between the GT processor and the interface boards are made possible through the DSPLINK interface of the DSP board [11] using a 50-pin ribbon cable and DSPLINK extender card. The extender card, which is inserted one of the other slots of the DSPLINK backplane cage, maps the fifty lines of DSPLINK to the 96-line backplane.

On the other end of the GT processor i/f board, there are two connections for the HSC: the HSC command interface; and the HSC data interface connection. A 26-pin ribbon cable and a 10-pin ribbon cable are used respectively to connect the GT processor i/f board to the corresponding connectors on the back of the HSC box. An additional 50-pin ribbon cable connector is located on the back of the HSC box and is used to connect the HSC to the frequency synthesizer. Details of the HSC configuration and interfaces are found in [3].

On the other end of the DIB, there are three connectors. A 26-pin ribbon cable connector is available to access one of the debug latches on the DIB. The second connector is a 9-pin male RS232 type connector. It is used to connect the data source (HP1645) to the DIB. Another 9-pin female RS232 type connector is used for the downlink synchronization reference serial communications link. Full details of the DIB connectors are found in [6].

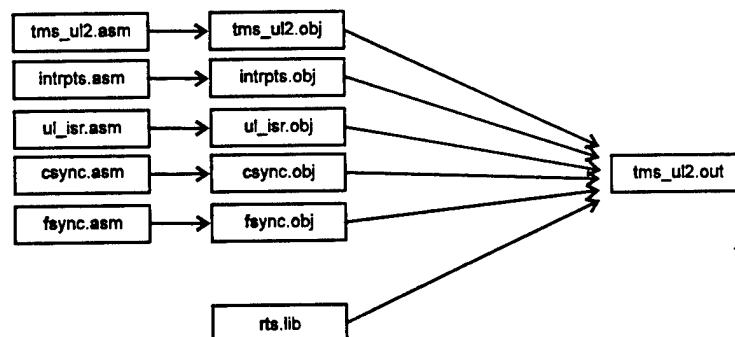
The serial port of the PC is also used in the GT simulator. The serial port of the PC is connected to the synchronization response return serial link.

## A1.2 Program files

In order to run the GT simulator, several files must be present. The files are listed below:

*tms\_ul2.out*  
*ulsync2.exe*  
*com.h*

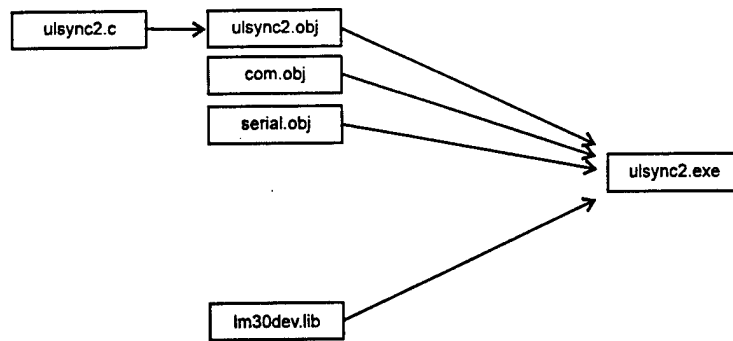
The GT simulator programs are compiled and executed in DOS. The *tms\_ul2.out* file is the output file from the TMS320C30 linker. The TMS320C30 linker creates the executable module by combining object files of compiled assembly programs and allocates sections to the DSP memory map. The GT simulator assembly routines being linked are shown in Fig. A.1. The file *rts.lib* is the run time library routines which are supplied by Spectrum Signal Processing Inc. The options and filenames for the linker are contained in a command file which is included in Section B7, of Appendix B.



**Fig. A.1 GT simulator TMS320C30 linker process**

Similarly, the *ulsync2.exe* file is the output of the C linker for the PC. The C linker combines the compiled host/user interface program with the TMS320C30 development library routines, *lm30dev.lib*, as well as the compiled serial communications routines. The definitions for the routines are found in the *com.h* header file. The result of the compiled routines is contained in two object files, *com.obj* and *serial.obj*. Fig. A.2 illustrates the flow of the C linker process for the PC.





**Fig. A.2 GT simulator host PC linker process**

## Appendix B: Software Listings

### B1. GT simulator host/user interface program

```
/* *****/
/*          Program Name:  ULSYNC2.C          */
/*          Author:       C. Tom              */
/*          Date Edited:   31 March 1998      */
/* *****/
/* THIS VERSION HAS A USER SELECTABLE OPTION FOR THE SERIAL COMMUNICATIONS */
/* ROUTINES.  USER IS PROMPTED AT THE BEGINNING OF PROGRAM.                */
/* *****/
/* Description                                                                */
/* *****/
/* PC program for GT simulator for uplink synchronization trials. Includes */
/* capability for remote operation through serial communications link        */
/* software.                                                                  */
/* Application                                                                */
/* *****/
/* 1. Available options to send CW at low, high, mid, 1/4, 3/4 band          */
/* 2. Available option to send CW at user specified frequency                */
/* 3. Available option to sweep through allocated BW (PN seq)                */
/* 4. Available option to go to RUN mode (frequency hopping)                 */
/* 5. Available option to enable/disable FRAME 0 detect (confirm DL)         */
/*    and adjust hop and frame counters.                                     */
/* 6. Available option to test CSYNC procedure                               */
/* 7. Available option to test FSYNC procedure                               */
/* *****/
/* ***** Include files *****/

#include <stdio.h>
#include <conio.h>
#include "com.h"

/* ***** TMS DPMEM address definitions *****/

#define BASEIO      0x290          /*base address of c30*/
#define BASEDP      0x30000L      /*base address of c30 dual port mem*/
#define C30DONE     BASEDP + 0L
#define MODE_AVAIL  BASEDP + 1L
#define MODE        BASEDP + 2L
#define MODE_ACK    BASEDP + 3L
#define F_AVAIL     BASEDP + 4L
#define F_VALUE     BASEDP + 5L
#define F_ACK       BASEDP + 6L
#define PHS_AVAIL   BASEDP + 7L
#define PHS_RNDED   BASEDP + 8L
#define PHS_ACK     BASEDP + 9L
#define DWELL_AVAIL BASEDP + 0xBL
#define DWELL_VAL   BASEDP + 0xCL
#define DWELL_ACK   BASEDP + 0xDL
#define VAL_RDY     BASEDP + 0xE
#define VAL_IDX     BASEDP + 0xF
#define VALUE       BASEDP + 0x10L
#define VAL_ACK     BASEDP + 0x11L
#define END_ASCII   BASEDP + 0x12L
#define STOP_MOD3   BASEDP + 0x13L
#define INCR_AVAIL  BASEDP + 0x14L
#define INCR_VAL    BASEDP + 0x15L
#define INCR_ACK    BASEDP + 0x16L
#define STOP_INT    BASEDP + 0x17L
#define RIS_DET     BASEDP + 0x18L
#define HYP_AVAIL   BASEDP + 0x19L
#define USER_HYP    BASEDP + 0x1AL
#define HYP_ACK     BASEDP + 0x1BL
#define DBUG_AVAIL  BASEDP + 0x1CL
#define DBUG_VALUE   BASEDP + 0x1DL
#define DBUG_ACK    BASEDP + 0x1EL
#define CHK_AVAIL   BASEDP + 0x1FL
#define CHK_VALUE    BASEDP + 0x20L
#define CHK_ACK     BASEDP + 0x21L
#define FRM_REF     BASEDP + 0x22L
#define CO_RESP     BASEDP + 0x23L
#define C1_RESP     BASEDP + 0x24L
#define FO_RESP     BASEDP + 0x25L
#define F1_RESP     BASEDP + 0x26L
#define RESP_AVAIL  BASEDP + 0x27L
#define RDY_4_RESP  BASEDP + 0x28L
#define RESP_ACK    BASEDP + 0x29L
#define RNG_XCDED   BASEDP + 0x2AL
```

```

#define CSYNC_OK          BASEDP + 0X2BL
#define CHK_HOP           BASEDP + 0X2CL
#define CHK_FRM           BASEDP + 0X2DL
#define LOG_END           BASEDP + 0X2EL
#define PLINE_FLAG        BASEDP + 0X2FL
#define DAT_AVAIL         BASEDP + 0X30L
#define INDEX             BASEDP + 0X31L
#define DAT_VALUE         BASEDP + 0X32L
#define DAT_ACK           BASEDP + 0X33L
#define END_FILE          BASEDP + 0X34L
#define TOO_MANY_HYPS     BASEDP + 0X35L
#define FRM_NOT_FOUND     BASEDP + 0X36L
#define FSTART_AVAIL      BASEDP + 0X40L
#define FSTART_FRM        BASEDP + 0X41L
#define FSTART_ACK        BASEDP + 0X42L
#define EST_AVAIL         BASEDP + 0X43L
#define FINE_EST          BASEDP + 0X44L
#define PHS_CHANGE        BASEDP + 0X45L
#define EST_ACK           BASEDP + 0X46L
#define NO_FSYNC          BASEDP + 0X47L
#define FSYNC_OK          BASEDP + 0X48L
#define UFLO_CDTN         BASEDP + 0X49L

/***** Miscellaneous constants *****/

#define DUAL               0          /*Memory type, p.48 in manual*/
#define ALL                1          /*Memory type, p.48 in manual*/
#define BITCLR             0L
#define BITSET             1L
#define maxlen             80
#define local              0
#define remote             1

#define MAX_HOP_NUM        0xfffffL  /* 16777215 for PN seq generator in HSC */
#define NCO_VAL            0x13A92A3L /* value of phs_rnded for NCO - default 192000 Hz*/

#define RESP_BUF_SIZ       100        /* Synch response buffer size */
#define STRING_LEN         220        /* Serial com message length */

/*External variable declaration*/
/*****/

int i, err, checkbit;
int ser_select = 0;
unsigned long check;
unsigned long lo_band, up_band, mid_band, oneq_band, threeq_band;
unsigned long base_f, hop_bw;
unsigned long flag, flag1;
int count, mode_of_op;
int stop_fsel;
unsigned long user_f, freq_val;
double hop_par;
float dwell_time;
unsigned long hop_incr;
unsigned long dwell_cycles;
FILE *f1p, *f2p, *f3p, *f4p, *f5p;
char fstring[maxlen], linetype, go_on_flg;
unsigned long idx;
long int data;
int int_mode = BITCLR;
int stop_PC = BITCLR;
int key_ret = BITCLR;
int msg_ret = BITCLR;
int proc_ret = BITCLR;
int operation;
int mlocal, ndest;
char string[STRING_LEN];
long int hypothesis;
long int sync_iter;
long int chk_point;
long int hop_b4edge, frm_b4edge;
long int command_given = BITCLR;
char c_resp_arr[RESP_BUF_SIZ][STRING_LEN];
long int f_resp_arr[RESP_BUF_SIZ];
int scmp_ret = BITCLR;
long int resp_rd_ptr = 0;
long int resp_wr_ptr = 0;
long int frame, c0_crsp, c1_crsp, c0_frsp, c1_frsp;
long int debug_entry;
unsigned long frm_ref_4_fine;
long int cum_fine_resp = 0;
int fine_cnt = 0;
long int f_est_avg, phs_change_4_nco;
float adjustment, flt_avg;
char u_input;

/* Subroutine declarations */

```

```

/*****/
void Init_DPMEM(void); /*initializes TMS dual port memory*/
unsigned long CTGet32Bit(int baseadr, long loc); /*alternate Get32Bit for DSP*/
int checkkey(int mdest); /*check and action key presses*/
int checkmsg(void); /*check for receive messages and others*/
void pabort(char *msg); /*print message, close file, exit*/
void Disp_menu(void);
void send_mode(int mode_sel);
void send_f(unsigned long f_val);
int proc_msg(int type, char mcontent[220]);
float Rd_value(char par[10]);

/****/
main()
/****/

{

/*****Load c30 program*****/

check = SelectBoard(BASEIO); /*Initialise c30*/
printf("\nReturn from SelectBoard = 0x %x h\n", check);
err = LoadObjectFile("tms_ul2.out"); /*Load c30 program*/
switch (err) {
case 0:
printf ("\nTMS program has been loaded successfully.\n");
break;
case 1:
printf ("\nERROR. Unable to open TMS file.\n");
exit(1);
break;
case 2:
printf ("\nERROR. Invalid address for TMS program.\n");
exit(1);
break;
default:
printf("\nError. Value = %i", err);
exit(1);
} /*end switch (err)*/

/*****Initialize TMS DP memory and synch response buffer in PC*****/

Init_DPMEM();

for (i=0; i < RESP_BUF_SIZ; i++)
{
strcpy(&c_resp_arr[i][0], "empty");
f_resp_arr[i] = 999;
} /* end for i */

/*****Start c30*****/

printf("\nSerial Comms routines enabled, enter '1', otherwise enter '0': ");
scanf("%i", &ser_select);
printf("\nSer_select value: %i ", ser_select);

Reset(); /*Reset DSP board*/
printf("\nReset issued to c30 board.\n");

/*****Download GT processor parameters*****/

if ((f5p=fopen("GTparam.dat", "r")) == NULL)
{
printf("\nError. Cannot open file GTPARAM.DAT. Aborting.");
exit(1);
} /*end if f5p*/
printf("\nFile opened for GT parameters.");
while (fgets(fstring, maxlen, f5p) != NULL)
{
printf("\r Reading next GT parameter. ");
sscanf(fstring, "%c %li %li", &linetype, &idx, &data);
if (linetype == 'd')
{
check = Put32Bit(INDEX, DUAL, idx);
check = Put32Bit(DAT_VALUE, DUAL, data);
check = Put32Bit(DAT_AVAIL, DUAL, BITSET);
flag = BITCLR;
while ((flag = CTGet32Bit(BASEIO, DAT_ACK)) != BITSET);
check = Put32Bit(DAT_ACK, DUAL, BITCLR);
} /*end if linetype*/
}

```

```

) /*end while fgets*/
printf("\nEnd of transferring GT parameters.");
check = Put32Bit(END_FILE, DUAL, BITSET);

/*****Send NCO initial condition to TMS*****/

check = Put32Bit(PHS_RNDED, DUAL, NCO_VAL);
check = Put32Bit(PHS_AVAIL, DUAL, BITSET);
flag1 = BITCLR;
while ((flag1 = CTGet32Bit(BASEIO, PHS_ACK)) != BITSET)
    printf("\rWaiting for TMS to ack transfer of NCO value.  ");
printf("\nNCO value transfer acknowledged.\n");
check = Put32Bit(PHS_ACK, DUAL, BITCLR);

/*****Read ASCII file with HSC parameters and transfer to TMS*****/

printf("\nReady to read HSC parameters.");
if ((f1p=fopen("hscinit.dat", "r"))==NULL){
    printf("\nError.  Cannot open input file for HSC parameters.");
    exit(1);
} /*end if f1p*/
printf("\nInput file opened for HSC parameters.");
while (fgets(fstring, maxlen, f1p) != NULL)
{
    printf("\rReading HSC parameters into TMS memory.");
    sscanf(fstring,"%c %li %lx", &linetype, &idx, &data);
    if (linetype == 'd')
    {
        check = Put32Bit(VAL_IDX, DUAL, idx);
        check = Put32Bit(VALUE, DUAL, data);
        check = Put32Bit(VAL_RDY, DUAL, BITSET);
        flag = BITCLR;
        while((flag=CTGet32Bit(BASEIO, VAL_ACK)) != BITSET);
        check = Put32Bit(VAL_ACK, DUAL, BITCLR);
    } /*end if linetype*/
} /*end while fgets*/
printf("\nEnd of HSC parameters ASCII file.");
check = Put32Bit(END_ASCII, DUAL, BITSET);
fclose(f1p);

/*****Read ASCII file with frequency values for SCITEQ or COMSTRON*****/

printf("\nReading frequency select values from ASCII file.");
if ((f2p=fopen("freq.dat", "r"))==NULL)
{
    printf("\nError.  Cannot open input file for frequency parameters.");
    exit(1);
} /*end if f2p*/
printf("\nInput file opened for frequency parameters.");
while (fgets(fstring, maxlen, f2p) != NULL)
{
    printf("\rReading frequency data.");
    sscanf(fstring,"%c %li %lx", &linetype, &idx, &data);
    if (linetype == 'd')
    {
        switch (idx)
        {
            case 0:
                lo_band = data;
                break;
            case 1:
                up_band = data;
                break;
            case 2:
                mid_band = data;
                break;
            case 3:
                oneq_band = data;
                break;
            case 4:
                threeq_band = data;
                break;
            case 5:
                stop_fsel = data;
                break;
            case 6:
                base_f = data;
                break;
            case 7:
                hop_bw = data;
                break;
        }
    }
}

```

```

        default:
            printf("\nError in data file index value.");
            exit(1);
        } /*end switch (idx)*/
    } /*end if linetype*/
} /*end while fgets*/
printf("\nEnd of freq.dat ASCII file.");
fclose(f2p);

/*****Initialize serial communications*****/
if (ser_select == 1)
{
    /*****Open all communications*****/
    if ((mlocal = open_com())==BAD_STATION) pabort("\nError in open_com.");
    printf("\nLocal station is %s\n", stnlstr(mlocal, string));

    /*****Select link to PL synch processor*****/
    if ((ndest=look_com("SYNC_PROC"))==BAD_STATION)
        pabort("\nBad station lookup");

    /*****Send message to remote terminal that local terminal is ready*****/
    printf("\nSending 'READY' message to remote terminal.");
    send_com(ndest,STATUS,"Local terminal ready.");
} /*end if (ser_select)*/

printf("\nLocal terminal ready.");

Disp_menu();

/*****Loop operation*****/
while (stop_PC == BITCLR)
{
    /***** Step 1: Check for local keyboard input *****/

    key_ret = checkkey(ndest);                /*Check for local keyboard input*/
    if (key_ret != 0)
    {
        if (key_ret == 1)                    /*regular exit*/
        {
            stop_PC = BITSET;
            printf("\nProgram halted by local user.");
            if (ser_select == 1)
                send_com(ndest,STATUS,"Program halted at local terminal.");
            break;
        } /*end if key_ret == 1*/
        else if (key_ret == 2)                /*exit slow hopping*/
        {
            printf("\nSlow hopping halted.");
            command_given = BITCLR;
            if (ser_select == 1)
                send_com(ndest,STATUS,"Slow hopping halted at local terminal.");
        } /*end else*/
    } /*end if key_ret*/

    /***** Step 2: Check for remote terminal input (if serial comms enabled)*****/

    if (ser_select == 1)
    {
        msg_ret = checkmsg();                /*Check for remote input*/
        switch (msg_ret)
        {
            case 0: /* normal exit */
                break;
            case 1: /* normal remote exit */
                stop_PC = BITSET;
                printf("\nProgram halted by remote user.");
                send_com(ndest, STATUS, "Program halted by remote terminal.");
                break;
            case 2: /* slow hopping halted by remote terminal */
                printf("\nSlow hopping halted by remote terminal.");
                send_com(ndest, STATUS, "Slow hopping halted by remote terminal.");
                break;
            case 3: /* Synch response buffer overflow */
                stop_PC = BITSET;
                printf("\nError. Synch resp buffer overflow. Abort.");
                send_com(ndest, STATUS, "Error. Synch resp buffer overflow. Abort.");
                break;
            case 4: /*Error in get_com found*/
                stop_PC = BITSET;
                printf("\nError in serial communications link detected.");
                break;
        }
    }
}

```

```

        default:
            printf("\nERROR. Invalid return value from check_msg subroutine.");
            exit(1);
        } /*end switch (msg_ret)*/
    } /*end if (ser_select)*/

/***** Step 3: Check for rising edge detect for DL synch *****/

    flag1 = BITCLR;
    if ((flag1 = CTGet32Bit(BASEIO,RIS_DET)) == BITSET)
    {
        hop_b4edge = CTGet32Bit(BASEIO,CHK_HOP);
        frm_b4edge = CTGet32Bit(BASEIO,CHK_FRM);
        printf("\nRISING EDGE DETECTED ON FRAME 0 LINE.");
        printf("\nValues of hop and frm counters before edge: %li %li", hop_b4edge, frm_b4edge);
        check = Put32Bit(RIS_DET, DUAL, BITCLR);
        printf("\nTo disable FRO detection, enter 'D' ");
    } /*end if flag1*/

/***** Step 4: Check for TMS response re: end of program *****/

    flag1 = BITCLR;
    if ((flag1=CTGet32Bit(BASEIO,C30DONE))==BITSET) /*check for TMS response*/
    {
        check = Put32Bit(C30DONE, DUAL, BITCLR);
        printf("\nTMS response received and cleared.");
        if (operation == local)
            printf("\nCommand executed. Another (y or n)? ");
        else if (operation == remote)
        {
            command_given = BITCLR;
            if (ser_select == 1)
                send_com(ndest, STATUS, "Op complete. Ready for command.");
            printf("\nRemote command executed.");
        } /*end else if*/
    } /*end if (flag1)*/

/***** Step 5: Check for transfer of next synch resp to TMS *****/

/* if (mode_of_op == 6) /* check for csynch mode */
/* {
/* if ((scmp_ret = strcmp(&c_resp_arr[resp_rd_ptr][0],"empty")) != 0) /*if buffer not empty*/
/* {
/* if ((flag1 = CTGet32Bit(BASEIO,RDY_4_RESP)) == BITSET)
/* {
/* sscanf(&c_resp_arr[resp_rd_ptr][0], "%i %i %i %i %i", &frame, &c0_crsp,
/* &c1_crsp, &c0_frsp, &c1_frsp);
/* check = Put32Bit(FRM_REF, DUAL, frame);
/* check = Put32Bit(C0_RESP, DUAL, c0_crsp);
/* check = Put32Bit(C1_RESP, DUAL, c1_crsp);
/* check = Put32Bit(F0_RESP, DUAL, c0_frsp);
/* check = Put32Bit(F1_RESP, DUAL, c1_frsp);
/* check = Put32Bit(RESF_AVAIL, DUAL, BITSET);
/* check = Put32Bit(RDY_4_RESP, DUAL, BITCLR);
/* strcpy(&c_resp_arr[resp_rd_ptr][0],"empty"); /*clear buffer element*/
/* if (resp_rd_ptr == (RESP_BUF_SIZ - 1)) /*check for buffer rollover*/
/* resp_rd_ptr = 0;
/* else
/* resp_rd_ptr = resp_rd_ptr + 1;
/* } /*end if flag1*/
/* } /*end if c_resp_arr*/
/* } /*end if mode_of_op == 6 */
/* else
/* {
/* if (mode_of_op == 7) /* check for fsynch mode */
/* {
/* if (f_resp_arr[resp_rd_ptr] != 999)
/* {
/* f_est_avg = f_resp_arr[resp_rd_ptr]; /* retrieve average fine estimate */
/* f_resp_arr[resp_rd_ptr] = 999; /* clear buffer element */
/* if (resp_rd_ptr == (RESP_BUF_SIZ - 1)) /* update pointer */
/* resp_rd_ptr = 0;
/* else
/* resp_rd_ptr = resp_rd_ptr + 1;
/* phs_change_4_nco = f_resp_arr[resp_rd_ptr]; /* retrieve phase change */
/* f_resp_arr[resp_rd_ptr] = 999; /* clear buffer element */
/* if (resp_rd_ptr == (RESP_BUF_SIZ - 1)) /* update pointer */
/* resp_rd_ptr = 0;
/* else
/* resp_rd_ptr = resp_rd_ptr + 1;
/* check = Put32Bit(FINE_EST, DUAL, f_est_avg);
/* check = Put32Bit(PHS_CHANGE, DUAL, phs_change_4_nco);
/* check = Put32Bit(EST_AVAIL, DUAL, BITSET);
/* } /*end if f_resp_arr*/
/* } /*end if mode_of_op == 7) */
/* } /* end else */

```

```

/***** Step 6: Check for proper transfer of synch resp to TMS, goes with Step 5 *****/
/*      if (mode_of_op == 6)
/*      {
/*          if ((flag1 = CTGet32Bit(BASEIO,RESP_ACK)) == BITSET)
/*          {
/*              check = Put32Bit(RESP_ACK, DUAL, BITCLR);
/*              check = Put32Bit(RDY_4_RESP, DUAL, BITSET);
/*          } /*end if flag1*/
/*      } /*end if mode_of_op == 6 */
/*      else
/*      {
/*          if (mode_of_op == 7)
/*          {
/*              if ((flag1 = CTGet32Bit(BASEIO,EST_ACK)) == BITSET)
/*              check = Put32Bit(EST_ACK, DUAL, BITCLR);
/*          } /*end if mode_of_op == 7 */
/*      } /*end else */

/***** Step 7:  Check for search range exceeded during CSYNC          *****/
/*****          Check for CSYNC_OK condition                        *****/
/*****          Check for pipeline overflow in synch response buffer *****/
/*****          Check for hyp_log overflow                          *****/
/*****          Check for FRM_NOT_FOUND when retrieving hypothesis *****/
/*****          Check for FSYNC_OK condition                      *****/
/*****          Check for NO_FSYNC condition - non-convergence     *****/
/*****          Check for FSTART_FRM for fine sync process         *****/

flag1 = BITCLR;
if ((flag1 = CTGet32Bit(BASEIO,RNG_XCDED)) == BITSET)
{
    check = Put32Bit(RNG_XCDED, DUAL, BITCLR);
    printf("\nSearch range exceeded encountered.");
    printf("\nDo you want to go back to DL sync? (y/n) ");
    scanf("%1s", &u_input);
    switch (u_input)
    {
        case 'Y':
        case 'y':
            int_mode = BITSET;
            mode_of_op = 4;
            send_mode(mode_of_op);
            printf("\nEnabling FR0 detection again.");
            if (ser_select == 1)
                send_com(ndest, STATUS, "LOC - back to FR0 detect");
            command_given = BITSET;
            break;
        case 'N':
        case 'n':
            stop_PC = BITSET;
            printf("\nExiting program now");
            break;
        default:
            printf("\nInvalid response. Aborting program.");
            stop_PC = BITSET;
            break;
    } /*end switch u_input*/
} /*end if ...RNG_XCDED*/
else /**1**/
{
    flag1 = BITCLR;
    if ((flag1 = CTGet32Bit(BASEIO,CSYNC_OK)) == BITSET)
    {
        check = Put32Bit(CSYNC_OK, DUAL, BITCLR);
        printf("\nCoarse synch achieved.");
        printf("\nDo you want to go to fine sync? (y/n) ");
        scanf("%1s", &u_input);
        switch (u_input)
        {
            case 'Y':
            case 'y':
                int_mode = BITSET;
                mode_of_op = 7;
                send_mode(mode_of_op);
                printf("\nProceeding to fine sync.");
                if (ser_select == 1)
                    send_com(ndest, STATUS, "LOC - going to fine sync");
                command_given = BITSET;
                break;
            case 'N':
            case 'n':
                stop_PC = BITSET;
                printf("\nExiting program now");
                break;
            default:

```



```

        printf("\nInvalid response. Aborting program.");
        stop_PC = BITSET;
        break;
    } /*end switch u_input*/
} /*end if ...CSYNC_OK*/
else /**2**/
{
    flag1 = BITCLR;
    if ((flag1 = CTGet32Bit(BASEIO,PLINE_FLAG)) == BITSET)
    {
        check = Put32Bit(PLINE_FLAG, DUAL, BITCLR);
        printf("\nSynch response buffer pipeline overflow in TMS. Exiting program.");
        stop_PC = BITSET;
    } /*end if flag1 ... PLINE_FLAG*/
    else /**3**/
    {
        flag1 = BITCLR;
        if ((flag1 = CTGet32Bit(BASEIO,TOO_MANY_HYPS)) == BITSET)
        {
            check = Put32Bit(TOO_MANY_HYPS, DUAL, BITCLR);
            printf("\nHypothesis buffer overflow. Exiting program.");
            stop_PC = BITSET;
        } /*end if flag1 ... TOO_MANY_HYPS*/
        else /**4**/
        {
            flag1 = BITCLR;
            if ((flag1 = CTGet32Bit(BASEIO,FRM_NOT_FOUND)) == BITSET)
            {
                check = Put32Bit(FRM_NOT_FOUND, DUAL, BITCLR);
                printf("\nUnable to retrieve hypothesis to verify coarse synch. Aborting.");
                stop_PC = BITSET;
            } /*end if flag1 ... FRM_NOT_FOUND*/
            else /**5**/
            {
                flag1 = BITCLR;
                if ((flag1 = CTGet32Bit(BASEIO, FSTART_AVAIL)) == BITSET)
                {
                    frm_ref_4_fine = CTGet32Bit(BASEIO,FSTART_FRM);
                    check = Put32Bit(FSTART_AVAIL, DUAL, BITCLR);
                    check = Put32Bit(FSTART_ACK, DUAL, BITSET);
                } /*end if flag1 ... FSTART_AVAIL*/
                else /**6**/
                {
                    flag1 = BITCLR;
                    if ((flag1 = CTGet32Bit(BASEIO, FSYNC_OK)) == BITSET)
                    {
                        check = Put32Bit(FSYNC_OK, DUAL, BITCLR);
                        printf("\nFine synch achieved. Ready to transmit data. \n\n");
                        stop_PC = BITSET;
                        /* TEST ONLY */
                    } /*end if flag1 ... FSYNC_OK*/
                    else /**7**/
                    {
                        flag1 = BITCLR;
                        if ((flag1 = CTGet32Bit(BASEIO, NO_FSYNC)) == BITSET)
                        {
                            check = Put32Bit(NO_FSYNC, DUAL, BITCLR);
                            printf("\nFine estimates not converging.");
                            printf("\nDo you want to go back to coarse sync? (y/n) ");
                            scanf("%c", &u_input);
                            switch (u_input)
                            {
                                case 'Y':
                                case 'y':
                                    mode_of_op = 6;
                                    send_mode(mode_of_op);
                                    hypothesis = 0;
                                    check = Put32Bit(USER_HYP, DUAL, hypothesis);
                                    check = Put32Bit(HYP_AVAIL, DUAL, BITSET);
                                    flag = BITCLR;
                                    while ((flag1 = CTGet32Bit(BASEIO, HYP_ACK)) == BITCLR)
                                        printf("\rAwaiting hyp acknowledge");
                                    check = Put32Bit(HYP_ACK, DUAL, BITCLR);
                                    if (ser_select == 1)
                                        send_com(ndest, STATUS, "LOC - going back to csync");
                                    break;
                                case 'N':
                                case 'n':
                                    printf("\nExiting program");
                                    stop_PC = BITSET;
                                    break;
                                default:
                                    printf("\nInvalid response. Exiting program now.");
                                    stop_PC = BITSET;
                                    break;
                            }
                        } /*end switch*/
                    } /*end if flag1 ... NO_FSYNC*/
                } /*end else *7* */
            }
        }
    }
}

```

```

        } /*end else *6* */
    } /*end else *5* */
    } /*end else *4* */
    } /*end else *3* */
    } /*end else *2* */
    } /*end else *1* */

/***** Step 8: Check for data underflow in GT processor i/f board *****/
    flag1 = BITCLR;
    if ((flag1 = CTGet32Bit(BASEIO,UFOLO_CDTN)) == BITSET)
    {
        printf("\nUnderflow condition received. Aborting program.");
        stop_PC = BITSET;
    } /*end if flag UFOLO_CDTN */

} /*end while (stop_PC)*/

/***** Disable TMS interrupts before exiting *****/

/*check = Put32Bit(STOP_INT, DUAL, BITSET);
*flag1 = BITSET;
*while ((flag1=CTGet32Bit(BASEIO,STOP_INT))==BITSET); */

if (ser_select == 1)
{
    while (ready_com(ndest)!=0)          /*wait for transmit buffer to be ready*/
        checkmsg();
    send_com(ndest, STATUS, "End of GT program.");
    while (chk_time(0) != 0);
    close_com();
} /*end if (ser_select)*/

printf("\nEnd of C program.");

} /*end of main*/

/*****/
unsigned long CTGet32Bit(int baseadr, long loc)
/*****/
{
    unsigned int low;
    unsigned int high;
    int C30port,commreg,hicommreg;
    unsigned long total;

    C30port=baseadr;
    commreg= C30port + 0;
    hicommreg= C30port + 2;
    SetAddr(loc);
    CntrDis();
    low=inpw(commreg);
    high=inpw(hicommreg);
    total = ((long)high<<16) + low;
    return(total);

} /*end CTGet32Bit()*/

/*****/
void Disp_menu()
/*****/
{
    printf("\n*****");
    printf("\n*          CW test - 7-11 April 1997          *");
    printf("\n*          GT Synch Processor Menu              *");
    printf("\n*");
    printf("\n* Enter one of the following:                    *");
    printf("\n*");
    printf("\n* 'L' : Go to lower edge of hop BW                *");
    printf("\n* 'U' : Go to upper edge of hop BW                *");
    printf("\n* 'M' : Go to middle of hop BW                  *");
    printf("\n* 'Q' : Go to one quarter mark of hop BW          *");
    printf("\n* 'T' : Go to three quarter mark of hop BW        *");
    printf("\n* 'F' : Go to specific frequency                  *");
    printf("\n* 'R' : Go to RUN mode                            *");
    printf("\n* 'S' : Slowly cycle through hop BW                *");
    printf("\n* 'E' : Enable interrupt/FRO detection            *");
    printf("\n* 'D' : Disable interrupt/FRO detection            *");
    printf("\n* 'C' : Coarse synchronization test                *");
    printf("\n* 'W' : Fine synchronization test                *");
    printf("\n* 'X' : Exit program or stop slow hopping (option 'S') *");
    if (int_mode == BITCLR)
        printf("\n* TMS interrupts/FRO detection is currently disabled. *");
    else
        printf("\n* TMS interrupts/FRO detection is currently enabled. *");
    printf("\n*****");
}

```

```

printf("\n\nEnter selection:  ");
return;
} /*end Disp_menu()*/

/*****
int checkkey(int dest)
*****/
{
    int c;                /*character from keyboard*/
    unsigned long user_f, freq_val;

if (kbhit() !=0)          /*is a key pressed?*/
{
    operation = local;
    c = getch();
    switch (command_given)
    {
        case 0:
            switch (c)
            {
                case 'L':
                case 'l':
                    mode_of_op = 1;
                    send_mode(mode_of_op);
                    freq_val = lo_band;
                    send_f(freq_val);
                    if (ser_select == 1)
                        send_com(ndest, STATUS, "LOC - go low command sent.");
                    command_given = BITSET;
                    break;
                case 'U':
                case 'u':
                    mode_of_op = 1;
                    send_mode(mode_of_op);
                    freq_val = up_band;
                    send_f(freq_val);
                    if (ser_select == 1)
                        send_com(ndest, STATUS, "LOC - go high command sent.");
                    command_given = BITSET;
                    break;
                case 'M':
                case 'm':
                    mode_of_op = 1;
                    send_mode(mode_of_op);
                    freq_val = mid_band;
                    send_f(freq_val);
                    if (ser_select == 1)
                        send_com(ndest, STATUS, "LOC - go mid command sent.");
                    command_given = BITSET;
                    break;
                case 'Q':
                case 'q':
                    mode_of_op = 1;
                    send_mode(mode_of_op);
                    freq_val = oneq_band;
                    send_f(freq_val);
                    if (ser_select == 1)
                        send_com(ndest, STATUS, "LOC - go 1/4 command sent.");
                    command_given = BITSET;
                    break;
                case 'T':
                case 't':
                    mode_of_op = 1;
                    send_mode(mode_of_op);
                    freq_val = threq_band;
                    send_f(freq_val);
                    if (ser_select == 1)
                        send_com(ndest, STATUS, "LOC - go 3/4 command sent.");
                    command_given = BITSET;
                    break;
                case 'F':
                case 'f':
                    printf("\nEnter frequency in 100Hz:  ");
                    scanf("%li", &user_f);
                    mode_of_op = 1;
                    send_mode(mode_of_op);
                    freq_val = user_f;
                    send_f(freq_val);
                    if (ser_select == 1)
                        send_com(ndest, STATUS, "LOC - user freq command sent.");
                    command_given = BITSET;
                    break;
                case 'R':
                case 'r':
                    mode_of_op = 2;
                    send_mode(mode_of_op);
                    printf("\nGoing to RUN mode.");
            }
        }
    }
}

```

```

        if (ser_select == 1)
            send_com(ndest, STATUS, "LOC - RUN command sent.");
        command_given = BITSET;
        break;
case 'S':
case 's':
    mode_of_op = 3;
    printf("\nEntry changes mode_of_op to : %i", mode_of_op);
    send_mode(mode_of_op);
    printf("\nPlease enter dwell in seconds: ");
    scanf("%f", &dwell_time);
    printf("\nPlease enter hop increment 0-16777215: ");
    scanf("%li", &hop_incr);
    dwell_cycles = (unsigned long)((dwell_time/60.0e-9) + 0.5);
    check = Put32Bit(DWELL_VAL, DUAL, dwell_cycles);
    check = Put32Bit(DWELL_AVAIL, DUAL, BITSET);
    flag1 = BITCLR;
    while ((flag1 = CTGet32Bit(BASEIO, DWELL_ACK)) != BITSET)
        printf("\rWaiting for TMS to ack receipt of dwell.  ");
    printf("\ndwell transferred.\n");
    check = Put32Bit(DWELL_ACK, DUAL, BITCLR);
    check = Put32Bit(INCR_VAL, DUAL, hop_incr);
    check = Put32Bit(INCR_AVAIL, DUAL, BITSET);
    flag1 = BITCLR;
    while ((flag1 = CTGet32Bit(BASEIO, INCR_ACK)) != BITSET);
    printf("\nhop increment transferred.\n");
    check = Put32Bit(INCR_ACK, DUAL, BITCLR);
    printf("\nTo stop slow hopping through BW, type 'X' ");
    if (ser_select == 1)
        send_com(ndest, STATUS, "LOC - sweep command sent.");
    command_given = BITSET;
    break;
case 'D':
case 'd':
    int_mode = BITCLR;
    mode_of_op = 5;
    send_mode(mode_of_op);
    printf("\nDisabling FR0 detection.");
    if (ser_select == 1)
        send_com(ndest, STATUS, "LOC - disabling FR0 detection.");
    command_given = BITSET;
    break;
case 'E':
case 'e':
    int_mode = BITSET;
    mode_of_op = 4;
    send_mode(mode_of_op);
    printf("\nEnabling FR0 detection.");
    if (ser_select == 1)
        send_com(ndest, STATUS, "LOC - enabling FR0 detection.");
    command_given = BITSET;
    break;
case 'C':
case 'c':
    mode_of_op = 6;
    send_mode(mode_of_op);
    printf("\nSync probes are transmitted starting at hop 288 by default.");
    printf("\nPlease enter desired hypothesis offset or 0 if none: ");
    scanf("%li", &hypothesis);
    printf("\nHyps will be at 0,1,-1,2,-2,...from hop 288 \\\(if no offset\\)");
    check = Put32Bit(USER_HYP, DUAL, hypothesis);
    check = Put32Bit(HYP_AVAIL, DUAL, BITSET);
    printf("\nHYP_AVAIL signal sent to TMS.");
    flag1 = BITCLR;
    while ((flag1=CTGet32Bit(BASEIO, HYP_ACK)) == BITCLR)
        printf("\rWaiting for hypothesis acknowledge.  ");
    printf("\nHypothesis transferred.");
    check = Put32Bit(HYP_ACK, DUAL, BITCLR);
    if (ser_select == 1)
        send_com(ndest, STATUS, "LOC - coarse synch initiated.");
    command_given = BITSET;
    break;
case 'W':
case 'w':
    mode_of_op = 7;
    send_mode(mode_of_op);
    printf("\nAttempting fine synch. \n");
    if (ser_select == 1)
        send_com(ndest, STATUS, "LOC - fine synch initiated");
    command_given = BITSET;
    break;
case 'X':
case 'x':
    if (mode_of_op == 3)
    {
        check = Put32Bit(STOP_MOD3, DUAL, BITSET);
        flag1 = BITSET;
    }

```

```

        while ((flag1 = CTGet32Bit(BASEIO, STOP_MOD3)) != BITCLR);
        printf("\nStopped TMS in mode3");
        mode_of_op = 0;
        return 2;
    }
    else
        return 1;
default:
    printf("\nInvalid entry. Try again.");
    Disp_menu();
    break;
} /*end switch (c), case 0 for command given*/
break;
case 1:
    switch(c)
    {
        case 'Y':
        case 'y':
            Disp_menu();
            command_given = BITCLR;
            if (ser_select == 1)
                send_com(ndest, STATUS, "LOC - Ready for next command.");
            break;
        case 'N':
        case 'n':
            if (ser_select == 1)
                send_com(ndest, STATUS, "LOC - user's stopping program.");
            return 1;
        case 'X':
        case 'x':
            if (mode_of_op == 3)
            {
                check = Put32Bit(STOP_MOD3, DUAL, BITSET);
                flag1 = BITSET;
                while ((flag1 = CTGet32Bit(BASEIO, STOP_MOD3)) != BITCLR);
                printf("\nStopped TMS in mode3");
                mode_of_op = 0;
                if (ser_select == 1)
                    send_com(ndest, STATUS, "LOC - exit slow hopping command.");
                return 2;
            }
            else
                return 1;
        default:
            printf("\nInvalid entry.");
            printf("\nPlease enter, y, n, or x to exit immediately");
            break;
    } /*end switch (c), case 1 for command given*/
    break;
} /*end switch(command_given)*/
} /*end if (kbhit())*/
return 0;
} /*end checkkey(dest)*/

/*****/
int checkmsg()
/*****/
{
    int mstat;           /*message status - valid, error, or quit*/
    int mtype;           /*message type number*/
    int mfrom;           /*message from station number*/
    char mdata[220];     /*message data*/
    char string[220];    /*message string buffer used to name, type, or error*/

    mstat = get_com(&mtype, &mfrom, mdata);
    if (mstat == VALID_MSG)
    {
        printf("\nMessage received from %s ", stnstr(mfrom, string));
        printf("\n(%s):  \"%s\"", messtr(mtype, string), mdata);
        proc_ret = proc_msg(mtype, mdata);
        return(proc_ret);
    } /*end if mstat*/
    else if (mstat == COMM_ERR)
    {
        printf("\n--Comm error with %s:  %s", stnstr(mfrom, string), mdata);
    } /*end else if mstat==COMM_ERR*/
    else if (mstat == QUIT)
    {
        if (mtype == TOTAL)
        {
            printf("\nToo many communication errors.");
            return 4;
        } /*end if mtype*/
        else if (mtype == CONSEC)
        {
            printf("\nToo many consecutive communication errors with %s",
                stnstr(mfrom, string));
        }
    }
}

```

```

        return 4;
    }
    else if (mtype == BREAK)
    {
        printf("\nBreak detected.");
    } /*end if mtype==BREAK*/
    return 1;
} /*end if mstat==QUIT*/
return 0;

} /*end checkmsg()*/

/*****/
void pabort(char *msg)
/*****/
{
    printf("\n%s" , msg);
    close_com();
    exit(0);
} /*end pabort()*/

/*****/
void send_mode(int mode_sel)
/*****/
{
    check = Put32Bit(MODE, DUAL, (long)mode_sel);    /*Transfer mode*/
    check = Put32Bit(MODE_AVAIL, DUAL, BITSET);      /*Signal TMS that mode is avail*/
    flag1 = BITCLR;
    while ((flag1 = CTGet32Bit(BASEIO, MODE_ACK)) != BITSET) {
        if (operation == local)
            printf("\rWaiting for TMS to ack mode transfer, flag1: %lx      ", flag1);
    } /*end while flag1*/
    if (operation == local) {
        printf("\nMode transfer acknowledged.");
    } /*end if operation*/
    check = Put32Bit(MODE_ACK, DUAL, BITCLR);
    return;
} /*end send_mode(mode_sel)*/

/*****/
void send_f(unsigned long f_val)
/*****/
{
    /*****Compute appropriate value for LD_HOP in HSC*****/
    hop_par = ( ( (double)(f_val - base_f) / (double)hop_bw ) * (double)MAX_HOP_NUM ) + 0.5;
    check = Put32Bit(F_VALUE, DUAL, (unsigned long)hop_par);

    check = Put32Bit(F_AVAIL, DUAL, BITSET);          /*signal TMS that frequency is avail*/
    flag1 = BITCLR;
    while ((flag1 = CTGet32Bit(BASEIO, F_ACK)) != BITSET) {
        if (operation == local)
            printf("\rWaiting for TMS to ack freq transfer, flag1: %lx      ", flag1);
    } /*end while flag1*/
    if (operation == local) {
        printf("\nFrequency select transfer acknowledged.\n");
    } /*end if operation*/
    check = Put32Bit(F_ACK, DUAL, BITCLR);
    return;
} /*end send_f(f_val)*/

/*****/
int proc_msg(int type, char mcontent[220])
/*****/
{
    char cmd[20], param1[20], param2[20];

    operation = remote;
    if (type == COMMAND)
    {
        /***** Break up message string into separate fields *****/

        sscanf(mcontent, "%s %s %s", cmd, param1, param2);

        if (strcmp(cmd, "Set_freq", 8)==0)
        {
            mode_of_op = 1;
            send_mode(mode_of_op);
            freq_val = (long int)Rd_value(param1);
            send_f(freq_val);
            printf("\nRemote user frequency sent to TMS.");
            command_given = BITSET;
        } /*end if strcmp = Set_freq*/
        else if (strcmp(cmd, "Go_to_run", 9)==0)
        {

```

```

mode_of_op = 2;
send_mode(mode_of_op);
printf("\nRemote RUN command sent to TMS.");
command_given = BITSET;
} /*end if strcmp = go_to_run*/
else if (strcmp(cmd, "Dwell_hop", 9)==0)
{
mode_of_op = 3;
send_mode(mode_of_op);
dwell_time = Rd_value(param1);
dwell_cycles = (unsigned long)((dwell_time/60.0e-9) + 0.5);
check = Put32Bit(DWELL_VAL, DUAL, dwell_cycles);
check = Put32Bit(DWELL_AVAIL, DUAL, BITSET);
flag1 = BITCLR;
while ((flag1 = CTGet32Bit(BASEIO, DWELL_ACK)) != BITSET) /*wait*/ ;
check = Put32Bit(DWELL_ACK, DUAL, BITCLR);
hop_incr = (long int)Rd_value(param2);
check = Put32Bit(INCR_VAL, DUAL, hop_incr);
check = Put32Bit(INCR_AVAIL, DUAL, BITSET);
flag1 = BITCLR;
while ((flag1 = CTGet32Bit(BASEIO, INCR_ACK)) != BITSET) /*wait*/ ;
check = Put32Bit(INCR_ACK, DUAL, BITCLR);
printf("\nRemote slow hop command sent to TMS.");
command_given = BITSET;
} /*end if strcmp = dwell_hop*/
else if (strcmp(cmd, "Exit", 4)==0)
{
if (mode_of_op == 3)
{
check = Put32Bit(STOP_MOD3, DUAL, BITSET);
flag1 = BITSET;
while ((flag1 = CTGet32Bit(BASEIO, STOP_MOD3)) != BITCLR);
printf("\nRemote stop TMS in mode3");
mode_of_op = 0;
send_com(ndest, STATUS, "Slow hop terminated.");
command_given = BITCLR;
return 2;
}
else
return 1;
} /*end if strcmp = exit*/
else if (strcmp(cmd, "Enable_FR0", 10)==0)
{
mode_of_op = 4;
send_mode(mode_of_op);
printf("\nRemote enable FR0 detect sent to TMS.");
command_given = BITSET;
} /*end if strcmp = Enable_FR0*/
else if (strcmp(cmd, "Disable_FR0", 11)==0)
{
mode_of_op = 5;
send_mode(mode_of_op);
printf("\nRemote disable FR0 detect sent to TMS.");
command_given = BITSET;
} /*end if strcmp = Disable_FR0*/
else if (strcmp(cmd, "Go_2_csync", 10)==0)
{
mode_of_op = 6;
send_mode(mode_of_op);
}

/*****NEED TO ADD IN INPUT FOR HYPOTHESIS OFFSET, ETC...*****/

printf("\nRemote CSYNC command sent to TMS.");
command_given = BITSET;
} /*end if strcmp = Go_2_csync*/
else
printf("\nWrong command statement: %s", mcontent);
send_com(ndest, STATUS, "Unknown command.");
} /*end if type = COMMAND*/
else if (type == STATUS)
{
if (isdigit(mcontent[0]) != 0) /* find out if 1st is a digit */
{
printf("\nStatus message processed: %s", mcontent); /*print status message*/
sscanf(mcontent, "%li", &frame);
if (mode_of_op == 6) /* check for CSYNC mode */
{
if ((frame%4) == 0) /* check for frame a mult of 4 */
{
if ((scmp_ret = strcmp(&c_resp_arr[resp_wr_ptr][0], "empty")) == 0)
{
strcpy(&c_resp_arr[resp_wr_ptr][0], mcontent); /*store synch resp*/
sscanf(mcontent, "%li %li %li %li", &frame, &c0_crsp, &c1_crsp,
&c0_frsp, &c1_frsp);
printf("\nFrame number, coarse resp, fine resp: %li %li %li %li",
frame, c0_crsp, c1_crsp, c0_frsp, c1_frsp);
if (resp_wr_ptr == (RESP_BUF_SIZ - 1)) /*update pointer*/

```

```

        resp_wr_ptr = 0;
    else
        resp_wr_ptr = resp_wr_ptr + 1;
        printf("\nSynch response stored.");
    } /*end if scmp_ret*/
    else
    {
        printf("\nERROR. Overflow of synch response buffer in PC.");
        return 3;
    } /*end else*/
} /*end if frame*/
} /*end if mode_of_op == 6 */
else
{
    if (mode_of_op == 7)
    {
        if (((frame%4) == 0) && (frame >= frm_ref_4_fine))
        {
            sscanf(mcontent, "%li %li %li %li %li", &frame, &c0_crsp, &c1_crsp,
                &c0_frsp, &c1_frsp);
            cum_fine_resp = cum_fine_resp + c1_frsp; /*cumul estimates for user 1*/
            fine_cnt = fine_cnt + 1;
            if (fine_cnt == 10)
            {
                if (cum_fine_resp >= 0)
                    flt_avg = ((float)cum_fine_resp/10.0) + 0.5; /* calc avg fine est */
                else
                    flt_avg = ((float)cum_fine_resp/10.0) - 0.5;
                f_est_avg = (long int)flt_avg;
                cum_fine_resp = 0; /* reset fine resp accumulator */
                fine_cnt = 0; /* reset fine estimate received count */
                if (f_resp_arr[resp_wr_ptr] == 999)
                {
                    f_resp_arr[resp_wr_ptr] = f_est_avg; /* store avg fine est */
                    if (resp_wr_ptr == (RESP_BUF_SIZ - 1)) /* update pointer */
                        resp_wr_ptr = 0;
                    else
                        resp_wr_ptr = resp_wr_ptr + 1;
                } /* end if f_resp_arr */
            }
            else
            {
                printf("\nERROR. Overflow of synch response buffer.");
                return 3;
            } /* end else */

            adjustment = (float)f_est_avg/(62.5 * 320.0); /*COMPUTE NCO CHANGE NECESSARY*/
            phs_change_4_nco = (long int)((1.0+adjustment)*NCO_VAL);
            if (f_resp_arr[resp_wr_ptr] == 999)
            {
                f_resp_arr[resp_wr_ptr] = phs_change_4_nco; /* store phase change */
                if (resp_wr_ptr == (RESP_BUF_SIZ - 1)) /* update pointer */
                    resp_wr_ptr = 0;
                else
                    resp_wr_ptr = resp_wr_ptr + 1;
            } /*end if f_resp_arr */
            else
            {
                printf("\nERROR. Overflow in synch resp buffer.");
                return 3;
            } /* end else */
        } /* end if fine_cnt */
    } /*end if frame...*/
} /*end if mode_of_op == 7*/
} /*end if isdigit...*/
else
    printf("\nStatus message: %s ", mcontent); /*print status message*/
} /*end else if type*/
return 0;
} /* end proc_msg(mcontent)*/

/*****
float Rd_value(char par[20])
/*****/
{
    float number;

    /* Find out whether it's a digit */

    if (isdigit(par[0]) == 0) { /*first char is not a digit*/
        if (strncmp(par, "low", 3)==0)
            return((float)lo_band);
        else if (strncmp(par, "high", 4)==0)
            return((float)up_band);
        else if (strncmp(par, "mid", 3)==0)
            return((float)mid_band);
        else if (strncmp(par, "oneq", 4)==0)

```



```

        return((float)oneq_band);
    else if (strcmp(par, "threeq", 6)==0)
        return((float)threeq_band);
    else {
        return(0.0);
    } /*end else*/
} /*end if isdigit*/
else {
    sscanf(par, "%f", &number);
    return(number);
} /*end else*/

} /*end Rd_value subroutine*/

/*****/
void Init_DPMEM()
/*****/
{
    check = Put32Bit(C30DONE,DUAL,BITCLR);
    check = Put32Bit(MODE_AVAIL,DUAL,BITCLR);
    check = Put32Bit(MODE,DUAL,BITCLR);
    check = Put32Bit(MODE_ACK,DUAL,BITCLR);
    check = Put32Bit(F_AVAIL,DUAL,BITCLR);
    check = Put32Bit(F_VALUE,DUAL,BITCLR);
    check = Put32Bit(F_ACK,DUAL,BITCLR);
    check = Put32Bit(PHS_AVAIL,DUAL,BITCLR);
    check = Put32Bit(PHS_RNDED,DUAL,BITCLR);
    check = Put32Bit(PHS_ACK,DUAL,BITCLR);
    check = Put32Bit(DWELL_AVAIL,DUAL,BITCLR);
    check = Put32Bit(DWELL_VAL,DUAL,BITCLR);
    check = Put32Bit(DWELL_ACK,DUAL,BITCLR);
    check = Put32Bit(VAL_RDY,DUAL,BITCLR);
    check = Put32Bit(VAL_IDX,DUAL,BITCLR);
    check = Put32Bit(VALUE,DUAL,BITCLR);
    check = Put32Bit(VAL_ACK,DUAL,BITCLR);
    check = Put32Bit(END_ASCII,DUAL,BITCLR);
    check = Put32Bit(STOP_MOD3,DUAL,BITCLR);
    check = Put32Bit(INCR_AVAIL,DUAL,BITCLR);
    check = Put32Bit(INCR_VAL,DUAL,BITCLR);
    check = Put32Bit(INCR_ACK,DUAL,BITCLR);
    check = Put32Bit(STOP_INT,DUAL,BITCLR);
    check = Put32Bit(RIS_DET,DUAL,BITCLR);
    check = Put32Bit(HYP_AVAIL,DUAL,BITCLR);
    check = Put32Bit(USER_HYP,DUAL,BITCLR);
    check = Put32Bit(HYP_ACK,DUAL,BITCLR);
    check = Put32Bit(DBGU_AVAIL,DUAL,BITCLR);
    check = Put32Bit(DBGU_VALUE,DUAL,BITCLR);
    check = Put32Bit(DBGU_ACK,DUAL,BITCLR);
    check = Put32Bit(CHK_AVAIL,DUAL,BITCLR);
    check = Put32Bit(CHK_VALUE,DUAL,BITCLR);
    check = Put32Bit(CHK_ACK,DUAL,BITCLR);
    check = Put32Bit(FRM_REF,DUAL,BITCLR);
    check = Put32Bit(CO_RESP,DUAL,BITCLR);
    check = Put32Bit(C1_RESP,DUAL,BITCLR);
    check = Put32Bit(F0_RESP,DUAL,BITCLR);
    check = Put32Bit(F1_RESP,DUAL,BITCLR);
    check = Put32Bit(RES_AVAIL,DUAL,BITCLR);
    check = Put32Bit(RES_ACK,DUAL,BITCLR);
    check = Put32Bit(RDY_4_RESP,DUAL,BITSET);
    check = Put32Bit(RNG_XCDED,DUAL,BITCLR);
    check = Put32Bit(CSYNC_OK,DUAL,BITCLR);
    check = Put32Bit(CHK_HOP,DUAL,BITCLR);
    check = Put32Bit(CHK_FRM,DUAL,BITCLR);
    check = Put32Bit(LOG_END,DUAL,BITCLR);
    check = Put32Bit(PLINE_FLAG,DUAL,BITCLR);
    check = Put32Bit(DAT_AVAIL,DUAL,BITCLR);
    check = Put32Bit(INDEX,DUAL,BITCLR);
    check = Put32Bit(DAT_VALUE,DUAL,BITCLR);
    check = Put32Bit(DAT_ACK,DUAL,BITCLR);
    check = Put32Bit(END_FILE,DUAL,BITCLR);
    check = Put32Bit(TOO_MANY_HYPS,DUAL,BITCLR);
    check = Put32Bit(FRM_NOT_FOUND,DUAL,BITCLR);
    check = Put32Bit(FSTART_AVAIL,DUAL,BITCLR);
    check = Put32Bit(FSTART_FRM,DUAL,BITCLR);
    check = Put32Bit(FSTART_ACK,DUAL,BITCLR);
    check = Put32Bit(EST_AVAIL,DUAL,BITCLR);
    check = Put32Bit(FINE_EST,DUAL,BITCLR);
    check = Put32Bit(PHS_CHANGE,DUAL,BITCLR);
    check = Put32Bit(EST_ACK,DUAL,BITCLR);
    check = Put32Bit(NO_FSYNC,DUAL,BITCLR);
    check = Put32Bit(FSYNC_OK,DUAL,BITCLR);
    check = Put32Bit(UFLO_CDTN,DUAL,BITCLR);

} /*end Init_DPMEM subroutine*/

```

## B2. DSP main program

```

*****
*               Program Name: TMS_UL2.ASM               *
*               Author: C. Tom                          *
*               Date edited: 26 February 1998           *
* Description: This program performs uplink GT processor functions including *
*               transmission of CW tones, slow sweeping across entire BW,   *
*               detecting FR0 pulse for DL sync confirmation, transmitting  *
*               of coarse sync probes and achieving coarse synchronization. *
*               This version has the parameters for the GT processor        *
*               downloaded from an ASCII data file.      *
*****

**** Subroutine declarations ****

                .globl      CHG_COM_DISP
                .globl      CHOOSE_F
                .globl      COMMAND_CLK
                .globl      CRSE_SYNC
                .globl      Disableint
                .globl      DSPDLAYLP
                .globl      Enableint
                .globl      FINE_SYNC
                .globl      GT_ISR
                .globl      HSC_INIT
                .globl      Ld_param_HSC
                .globl      NRDY_low_loop
                .globl      Rd_GTparam
                .globl      Rd_HSCparam
                .globl      SLOW_HOP
                .globl      START_NCO

**** Miscellaneous constants ****

XFO_EN          .set        2h
XF_SET          .set        6h
XF_CLR          .set        0fffbh

BITCLR          .set        0
BITSET          .set        1

MODE1           .set        1           ;mode 1: CW frequency transmit
MODE2           .set        2           ;mode 2: RUN mode
MODE3           .set        3           ;mode 3: sweep alloc bw with dwell
MODE4           .set        4           ;mode 4: FR0 enable mode
MODE5           .set        5           ;mode 5: FR0 disable mode
MODE6           .set        6           ;mode 6: CSYNC mode
MODE7           .set        7           ;mode 7: FSYNC mode

F_STOP          .set        1234h
MASKL16         .set        0ffffh

VAR_BASE        .data
                .word        gt_vars

* Indices for gt_vars parameters array

NUM_HOP         .set        0
MAX_FRM         .set        1
MAX_HOP         .set        2
PRB_START       .set        3
SRCH_LIM        .set        4
TIMES_4_CONFM   .set        5
RESP_BUF_SIZ    .set        6
MIN_DET_2_VER    .set        7
NUM_RETRANSMITS .set        8
LIM_10          .set        9
MIN_4_CONV       .set        10
MAX_ATTEMPTS     .set        11

                .data
HSC_addr        .word        HSC_array
BITS0_N_1       .word        3h
BITS2_31        .word        0FFFFFFFCh

**** DSP initialization stuff ****

                .data
BEGSTACK        .word        809800h      ;Address of beginning of stack
Pribus          .set        808064h      ;Address of primarybus control register
Secbus          .set        808060h      ;Address of secondary bus control register

Pribusval       .set        800h
Secbusval       .set        0
ST_reg_init     .set        1800h        ;bit to clear/enable CACHE, disable OVM

```

\* N.B. After RESET, following registers are initialized to zero

\* ST - CPU status register  
 \* IE - CPU/DMA interrupt enable flags  
 \* IF - CPU interrupt flags  
 \* IOF - I/O flags

\*\*\*\* Data page pointers \*\*\*\*

INIPG	.set	0
DPMEMPG	.set	3h
BSSPG	.set	80h
DATAPG	.set	0h
DSPLNPKG	.set	80h
BUSPG	.set	80h

\*\*\*\* DPMEM addresses \*\*\*\*

DPBASE	.set	30000h
C30DONE	.set	DPBASE
MODE_AVAIL	.set	DPBASE + 1h
MODE	.set	DPBASE + 2h
MODE_ACK	.set	DPBASE + 3h
F_AVAIL	.set	DPBASE + 4h
F_VALUE	.set	DPBASE + 5h
F_ACK	.set	DPBASE + 6h
PHS_AVAIL	.set	DPBASE + 7h
PHS_RNDED	.set	DPBASE + 8h
PHS_ACK	.set	DPBASE + 9h
DWELL_AVAIL	.set	DPBASE + 08h
DWELL_VAL	.set	DPBASE + 0Ch
DWELL_ACK	.set	DPBASE + 0Dh
VAL_RDY	.set	DPBASE + 0Eh
VAL_IDX	.set	DPBASE + 0Fh
VALUE	.set	DPBASE + 10h
VAL_ACK	.set	DPBASE + 11h
END_ASCII	.set	DPBASE + 12h
STOP_MOD3	.set	DPBASE + 13h
INCR_AVAIL	.set	DPBASE + 14h
INCR_VAL	.set	DPBASE + 15h
INCR_ACK	.set	DPBASE + 16h
STOP_INT	.set	DPBASE + 17h
RIS_DET	.set	DPBASE + 18h

\* DPBASE+19h to DPBASE+2Bh, AND DPBASE+2Eh to DPBASE + 2Fh used in CSYNC.ASM

CHK_HOP	.set	DPBASE + 2Ch
CHK_FRM	.set	DPBASE + 2Dh
DAT_AVAIL	.set	DPBASE + 30h
INDEX	.set	DPBASE + 31h
DAT_VALUE	.set	DPBASE + 32h
DAT_ACK	.set	DPBASE + 33h
END_FILE	.set	DPBASE + 34h

\*\*\*\* BER i/f board addresses \*\*\*\*

CMD_BER	.set	800009h	;WRITE only
STAT_BER	.set	800009h	;READ only
BER_DAT_port	.set	800008h	;READ/WRITE

\*\*\*\* BER i/f board commands \*\*\*\*

BER_SWres	.set	8000h
BER_Dfault	.set	0h

\*\*\*\* GT i/f board addresses \*\*\*\*

COMMAND	.set	800004h	;WRITE only
STATUS	.set	800004h	;READ only
NCO_CMD	.set	800005h	;WRITE only
INTRPT_PORT	.set	800005h	;READ only
HSC_PORT	.set	800006h	;WRITE only
FSK_FRM	.set	800007h	;WRITE only

\*\*\*\* GT i/f board commands \*\*\*\*

SW_RES_GT	.set	1
Dfault_CMD	.set	5

\*\*\*\* HSC commands and miscellaneous \*\*\*\*

.data

```

HSC_endhop      .word      0ffffffh      ;16777215 (max hop number for HSC)

SYNC_BIT        .set      2              ;SYNC on bit D1 of GT status
NRDY_BIT        .set      4              ;bit D2 on GT status (DSPLINK)

STOP_HSC        .set      0
RUN_HSC         .set      8000h
CHG_IMMED       .set      0501h
CHG_HOP         .set      0500h
LD_LATCH        .set      0300h
LD_BASE         .set      030Ch
LD_BWSCALE      .set      0306h
LD_DOPF         .set      0310h
LD_FCSPACE      .set      030Ah
LD_FLAGS        .set      031Ah
LD_FSKCHAN      .set      0308h
LD_HOP          .set      0304h
LD_LOSCI        .set      0318h
LD_LOCOM        .set      0312h
LD_OFFSET       .set      030Eh
LD_TIMELO       .set      0314h
LD_TIMEHI       .set      0316h

ULSYNC_CMD      .set      500Bh          ;GO TO ULSYNC MODE, channel 1, bin 3
ULGO_BASE       .set      200h          ;BASE VALUE OF ULGO COMMAND FOR CSYNC

```

\*\*\*\* HSC parameter indices \*\*\*\*

```

BASE_L16        .set      0
BASE_H16        .set      1
BWSCALE_L16     .set      2
BWSCALE_H16     .set      3
HOP_BW_L16      .set      4
HOP_BW_H16      .set      5
DOPF_L16        .set      6
DOPF_H16        .set      7
FCSPACE_L16     .set      8
FCSPACE_H16     .set      9
FLAGS_L16       .set      10
FLAGS_H16       .set      11
LOSCI_L16       .set      12
LOSCI_H16       .set      13
LOCOM_L16       .set      14
LOCOM_H16       .set      15
OFFSET_L16      .set      16
OFFSET_H16      .set      17
TIMELO_L16      .set      18
TIMELO_H16      .set      19
TIMEHI_L16      .set      20
TIMEHI_H16      .set      21

```

\*\*\*\* NCO constants and commands \*\*\*\*

```

                .data
NCO_INIT        .float      1.92e5
NCO_const       .float      1.0737418e2
D0_D7mask       .word      0FFh
D8_D15mask      .word      0FF00h
D16_D23mask     .word      0FF0000h
D24_D31mask     .word      0FF000000h
D0_D7addr       .word      3000000h
D8_D15addr      .word      2000000h
D16_D23addr     .word      1000000h
D24_D31addr     .word      0h
Addr_Phase      .word      Phase
NCO_WRN_LO      .set      4h
NCO_STRB_HI     .set      7h

```

\*\*\*\* Reserve memory in .bss for variables \*\*\*\*

```

                .globl      op_mode
                .bss        op_mode,1
                .globl      LSB16
                .bss        LSB16,1
                .globl      MSB16
                .bss        MSB16,1
                .globl      f_select
                .bss        f_select,1
                .globl      HOP_PAR
                .bss        HOP_PAR,1
                .globl      temp_var
                .bss        temp_var,1
                .globl      NCO_CLK
                .bss        NCO_CLK,1
                .globl      Phase
                .bss        Phase,4

```

```

        .globl      phs_rnded
        .bss        phs_rnded,1
        .globl      original_phase
        .bss        original_phase,1
        .globl      hop_dwell
        .bss        hop_dwell,1
        .globl      HSC_array
        .bss        HSC_array,25
        .globl      counter
        .bss        counter,1
        .globl      tx_cnt
        .bss        tx_cnt,1
        .globl      gt_vars
        .bss        gt_vars,12

**** Variables defined elsewhere ****

        .globl      BER_stat
        .globl      hop_cnt
        .globl      frm_cnt
        .globl      prev_FR0
        .globl      chk_FR0_flg
        .globl      array_cnt
        .globl      hyp_used
        .globl      hyp_offset
        .globl      nxt_prb_frm
        .globl      act_prb_frm
        .globl      act_prb_hop
        .globl      start_timef
        .globl      start_timeh
        .globl      assigned_f
        .globl      tms_csync_rdy
        .globl      brst0_flg
        .globl      brst1_flg
        .globl      user_hyp_off
        .globl      iter_hyp
        .globl      first_prb
        .globl      prb_cmd
        .globl      coarse1
        .globl      coarse2
        .globl      ref_frame

*** Initialize RESET and interrupt service routine locations ****

        .sect      "VECTORS"

RESET      .word      START
INT0        .word      GT_ISR
INT1        .word      0
INT2        .word      0
INT3        .word      0
XINT0       .word      0
RINT0       .word      0
XINTA       .word      0
RINTA       .word      0
TINT0       .word      0
TINT1       .word      0
DINT        .word      0
            .space     20

**** Program begins here ****

        .text

START:
        LDI          INIPG,DP          ;set DP pointer to 0
        LDI          ST_reg_init,ST    ;clear/enable CACHE, disable OVM
        LDI          DATAPG,DP
        LDI          @BEGSTACK,SP      ;initialize SW stack pointer

        LDI          BUSPG,DP          ;initialize primary bus control register
        LDI          Pribusval,R4
        STI          R4,@Pribus
        LDI          Secbusval,R4      ;initialize secondary bus control register
        STI          R4,@Secbus

        LDI          DSPLNKPG,DP

        LDI          SW_RES_GT,R0      ;issue SW reset of GT i/f board
        LSH          16,R0
        STI          R0,@COMMAND
        CALL         DSPDLAYLP

        LDI          Dfault_CMD,R0
        LSH          16,R0
        STI          R0,@COMMAND
        CALL         DSPDLAYLP

```

```

        LDI        BER_SWres,R0        ;issue SW reset of BER i/f board
        LSH        16,R0
        STI        R0,@CMD_BER
        CALL       DSPDLAYLP

        LDI        BER_Dfault,R0
        LSH        16,R0
        STI        R0,@CMD_BER
        CALL       DSPDLAYLP

* Download parameters for GT processor

        CALL       Rd_GTparam

* Wait for clock frequency from PC

NCO_WAIT:    LDI        DPMEMPG,DP

        LDI        @PHS_AVAIL,R0
        CMPI       BITSET,R0
        BNE        NCO_WAIT

        LDI        BITCLR,R0
        STI        R0,@PHS_AVAIL

        LDI        @PHS_RNDED,R0
        LDI        BSSPG,DP
        STI        R0,@phs_rnded
        STI        R0,@original_phase

        LDI        BITSET,R0
        LDI        DPMEMPG,DP
        STI        R0,@PHS_ACK

        CALL       START_NCO

* Download initial parameters for HSC (Blue box)

        CALL       Rd_HSCparam

        CALL       HSC_INIT

        CALL       Ld_param_HSC

* Final initialization of variables

        LDI        DSPLNKPG,DP        ;make sure RF is on
        LDI        0,R0
        STI        R0,@FSK_FRM
        CALL       DSPDLAYLP

        LDI        DSPLNKPG,DP        ;read GT status to clear uflo bit
        LDI        @STATUS,R0
        CALL       DSPDLAYLP

        LDI        0,R0                ;clear flags for ISR
        LDI        BSSPG,DP
        STI        R0,@BER_stat
        STI        R0,@chk_FR0_flg
        STI        R0,@array_cnt
        STI        R0,@tms_csync_rdy

        LDI        800h,R0             ;initialize prev_FR0 to be "high"
        STI        R0,@prev_FR0

        LDI        222,R0             ;arbitrary
        STI        R0,@hop_cnt
        LDI        7,R0
        STI        R0,@frm_cnt

* Enable interrupts

        LDI        0,IF                ;clear any pending interrupts
        CALL       Enableint

NEXT_CMD:

* Wait for mode of operation from PC

        LDI        DPMEMPG,DP

MODE_WAIT:   LDI        @MODE_AVAIL,R0
        CMPI       BITSET,R0
        BEQ        GET_MODE
        LDI        @STOP_INT,R0
        CMPI       BITSET,R0
        BEQ        CLOSE_OUT
        B          MODE_WAIT

```

```

GET_MODE:
        LDI        BITCLR,R0        ;clear MODE_AVAIL location
        STI        R0,@MODE_AVAIL

        LDI        @MODE,R0        ;read MODE
        LDI        BSSPG,DP
        STI        R0,@op_mode

        LDI        BITSET,R0        ;acknowledge MODE transfer
        LDI        DPMEMPG,DP
        STI        R0,@MODE_ACK

* Put HSC in known state (STOP MODE)

        CALL        HSC_INIT

* Find out which mode

        LDI        BSSPG,DP
        LDI        @op_mode,R0
        CMPI       MODE1,R0
        BEQ        OP_1
        CMPI       MODE2,R0
        BEQ        OP_2
        CMPI       MODE3,R0
        BEQ        OP_3
        CMPI       MODE4,R0
        BEQ        OP_4
        CMPI       MODE5,R0
        BEQ        OP_5
        CMPI       MODE6,R0
        BEQ        OP_6
        CMPI       MODE7,R0
        BEQ        OP_7

OP_ERR:
        LDI        BSSPG,DP        ;Error. Synth goes to low edge of alloc BW
        LDI        0h,R0
        STI        R0,@LSB16
        LDI        0h,R0
        STI        R0,@MSB16
        CALL        CHG_COM_DISP
        B          RET_MODE
*****

OP_1:
        CALL        CHOOSE_F
        B          RET_MODE
*****

OP_2:
        LDI        DSPLNKPG,DP
        CALL        NRDY_low_loop

        LDI        CHG_HOP,R0        ;issue CHG_HOP to respond to rising edge of HCLK
        LSH        16,R0
        STI        R0,@HSC_PORT
        CALL        DSPDLAYLP

        CALL        NRDY_low_loop

        LDI        RUN_HSC,R0        ;issue RUN command
        LSH        16,R0
        STI        R0,@HSC_PORT
        CALL        DSPDLAYLP
        B          RET_MODE
*****

OP_3:
DWELL_WAIT:
        LDI        DPMEMPG,DP
        LDI        @DWELL_AVAIL,R0
        CMPI       BITSET,R0
        BNE        DWELL_WAIT

        LDI        BITCLR,R0
        STI        R0,@DWELL_AVAIL

        LDI        @DWELL_VAL,R0
        LDI        BSSPG,DP
        STI        R0,@hop_dwell

        LDI        BITSET,R0
        LDI        DPMEMPG,DP
        STI        R0,@DWELL_ACK

INCR_WAIT:
        LDI        DPMEMPG,DP
        LDI        @INCR_AVAIL,R0
        CMPI       BITSET,R0
        BNE        INCR_WAIT

```

```

        LDI        BITCLR,R0
        STI        R0,@INCR_AVAIL

        LDI        @INCR_VAL,R0
        LDI        BSSPG,DP
        STI        R0,@hop_incr

        LDI        BITSET,R0
        LDI        DPMEMPG,DP
        STI        R0,@INCR_ACK

        CALL       SLOW_HOP
        B          RET_MODE

*****
OP_4:
FR0_DET:
        LDI        BITSET,R0
        LDI        BSSPG,DP
        STI        R0,@chk_FR0_flg
        B          RET_MODE

*****
OP_5:
NO_FR0:
        LDI        BITCLR,R0
        LDI        BSSPG,DP
        STI        R0,@chk_FR0_flg
        LDI        DPMEMPG,DP      ;clear pending RIS_DET
        STI        R0,@RIS_DET
        B          RET_MODE

*****
OP_6:
        CALL       CRSE_SYNC
        B          NEXT_CMD      ;ONLY APPLIES TO MODE6 & MODE7
*****
OP_7:
        CALL       FINE_SYNC
        B          NEXT_CMD

*****
RET_MODE:
        LDI        DPMEMPG,DP
        LDI        BITSET,R0
        STI        R0,@C30DONE
        B          NEXT_CMD

CLOSE_OUT:
        CALL       Disableint
        LDI        DPMEMPG,DP
        LDI        BITCLR,R0
        STI        R0,@STOP_INT

STDBY:
        NOP
        B          STDBY      ;C30 waits here

*****
*          SUBROUTINES BEGIN HERE          *
*****

        .globl     CHG_COM_DISP

CHG_COM_DISP:
        PUSH       DP
        PUSH       R0
        PUSHF      R0

        LDI        DSPLNPKG,DP

        CALL       NRDY_low_loop      ;wait until HSC has processed command

        LDI        CHG_IMMED,R0      ;Change immediate command
        LSH        16,R0
        STI        R0,@HSC_PORT
        CALL       DSPDLAYLP

        CALL       NRDY_low_loop

        ;
        LDI        LD_LATCH,R0      ;Load Latch command
        LDI        LD_HOP,R0      ;Load Hop command
        LSH        16,R0
        STI        R0,@HSC_PORT
        CALL       DSPDLAYLP

```



```

CALL        NRDY_low_loop

LDI         BSSPG,DP
LDI         @LSB16,R0      ;16 LSB of COMSTRON synthesizer latch
LSH         16,R0
LDI         DSPLNKPG,DP
STI         R0,@HSC_PORT
CALL        DSPDLAYLP

CALL        NRDY_low_loop

LDI         BSSPG,DP
LDI         @MSB16,R0      ;16 MSB of COMSTRON synthesizer latch
LSH         16,R0
LDI         DSPLNKPG,DP
STI         R0,@HSC_PORT
CALL        DSPDLAYLP

CALL        NRDY_low_loop

POPF        R0
POP         R0
POP         DP

RETS                                ;end of CHG_COM_DISP subroutine
*****

        .globl        CHOOSE_F

CHOOSE_F:

PUSH        DP
PUSH        R0
PUSHF       R0
PUSH        R1
PUSHF       R1
PUSH        R2
PUSHF       R2
PUSH        R3
PUSHF       R3
PUSH        R4
PUSHF       R4
PUSH        R5
PUSHF       R5
PUSH        R6
PUSHF       R6

* Wait for frequency selected by PC

F_WAIT:    LDI         DPMEMPG,DP

LDI         @F_AVAIL,R0      ;wait until frequency selection is available
CMPI       BITSET,R0
BNE        F_WAIT

LDI         BITCLR,R0      ;clear F_AVAIL location
STI         R0,@F_AVAIL

LDI         @F_VALUE,R0      ;read F_VALUE
LDI         BSSPG,DP
STI         R0,@f_select

LDI         BITSET,R0      ;acknowledge transfer of F_VALUE
LDI         DPMEMPG,DP
STI         R0,@F_ACK

* Send appropriate frequency to HSC

* First, check that it's not an F_STOP command

LDI         BSSPG,DP
LDI         @f_select,R0      ; R0 <- f_select
CMPI       F_STOP,R0
BEQ        GET_OUT

* For now, value to be transferred to HSC computed by PC

AND         MASKL16,R0      ;store 16 LSB
STI         R0,@LSB16
LDI         @f_select,R0
LSH         -16,R0          ;store 16 MSB
AND         MASKL16,R0
STI         R0,@MSB16

CALL        CHG_COM_DISP

```

```

GET_OUT:
        POPF      R6
        POP      R6
        POPF      R5
        POP      R5
        POPF      R4
        POP      R4
        POPF      R3
        POP      R3
        POPF      R2
        POP      R2
        POPF      R1
        POP      R1
        POPF      R0
        POP      R0
        POP      DP

        RETS                      ;end of CHOOSE_F subroutine
*****

        .globl      COMMAND_CLK

        .globl      index
        .bss        index,1

        .text
COMMAND_CLK:
        PUSH      DP
        PUSH      R0
        PUSHF     R0
        PUSH      R1
        PUSHF     R1
        PUSH      R2
        PUSHF     R2
        PUSH      AR0
        PUSH      IRO

        LDI      BSSPG,DP
        LDI      0,R0
        LDI      R0,IRO
        STI      R0,@index

        LDI      DATAPG,DP
        LDI      @Addr_Phase,AR0

Phase_trans:
        LDI      **AR0(IRO),R1
        LDI      DSPLNKPG,DP
        STI      R1,@NCO_CMD
        CALL     DSPDLAYLP
        LDI      NCO_WRN_LO,R2
        LSH      16,R2
        STI      R2,@COMMAND
        CALL     DSPDLAYLP
        LDI      Dfault_CMD,R2
        LSH      16,R2
        STI      R2,@COMMAND
        CALL     DSPDLAYLP
        LDI      BSSPG,DP
        LDI      @index,R0
        ADDI     1,R0
        LDI      R0,IRO
        STI      R0,@index
        CMPI     4,R0
        BLT      Phase_trans

        LDI      DSPLNKPG,DP
        LDI      NCO_STRB_HI,R2
        LSH      16,R2
        STI      R2,@COMMAND
        CALL     DSPDLAYLP
        LDI      Dfault_CMD,R2
        LSH      16,R2
        STI      R2,@COMMAND
        CALL     DSPDLAYLP

        POP      IRO
        POP      AR0
        POPF     R2
        POP      R2
        POPF     R1
        POP      R1
        POPF     R0
        POP      R0
        POP      DP

```

```

                RETS                                ;end of COMMAND_CLK subroutine
*****

                .globl        DSPDLAYLP

DSPDLAYLP:
                PUSH          DP
                PUSH          R1
                PUSHF         R1

                LDI           0,R1

DLAY:
                ADDI          1,R1
                CMPI          5,R1
                BLT           DLAY

                POPF          R1
                POP           R1
                POP           DP

                RETS                                ;end of DSPDLAYLP subroutine
*****

                .globl        HSC_INIT

HSC_INIT:
                PUSH          DP
                PUSH          R0
                PUSHF         R0

                CALL          NRDY_low_loop          ;initialize HSC

                LDI           DSPLNKPG,DP

                LDI           STOP_HSC,R0           ;1st STOP command
                LSH           16,R0
                STI           R0,@HSC_PORT
                CALL          DSPDLAYLP

                CALL          NRDY_low_loop          ;wait until HSC has processed command

                LDI           STOP_HSC,R0           ;2nd STOP command
                LSH           16,R0
                STI           R0,@HSC_PORT
                CALL          DSPDLAYLP

                CALL          NRDY_low_loop          ;wait until HSC has processed command

                LDI           STOP_HSC,R0           ;3rd STOP command
                LSH           16,R0
                STI           R0,@HSC_PORT
                CALL          DSPDLAYLP

                POPF          R0
                POP           R0
                POP           DP

                RETS                                ;end of HSC_INIT subroutine
*****

                .globl        Ld_param_HSC

Ld_param_HSC:
                PUSH          DP
                PUSH          R0
                PUSHF         R0
                PUSH          AR0
                PUSH          IRO

                LDI           DATAPG,DP
                LDI           @HSC_addr,AR0

                LDI           DSPLNKPG,DP

                CALL          NRDY_low_loop          ;wait until HSC has processed command
                LDI           LD_BASE,R0           ;Load BASE command
                LSH           16,R0
                STI           R0,@HSC_PORT
                CALL          DSPDLAYLP

                CALL          NRDY_low_loop          ;wait until HSC has processed command
                LDI           BASE_L16,IRO         ;BASE_L16 transfer
                LDI           *+AR0(IRO),R0

```

LSH	16,R0	
STI	R0,@HSC_PORT	
CALL	DSPDLAYLP	
CALL	NRDY_low_loop	;wait until HSC has processed command
LDI	BASE_H16,IR0	;BASE_H16 transfer
LDI	*+AR0(IR0),R0	
LSH	16,R0	
STI	R0,@HSC_PORT	
CALL	DSPDLAYLP	
CALL	NRDY_low_loop	;wait until HSC has processed command
LDI	LD_BWSALE,R0	;Load BWSALE command
LSH	16,R0	
STI	R0,@HSC_PORT	
CALL	DSPDLAYLP	
CALL	NRDY_low_loop	;wait until HSC has processed command
LDI	BWSALE_L16,IR0	;BWSALE_L16 transfer
LDI	*+AR0(IR0),R0	
LSH	16,R0	
STI	R0,@HSC_PORT	
CALL	DSPDLAYLP	
CALL	NRDY_low_loop	;wait until HSC has processed command
LDI	BWSALE_H16,IR0	;BWSALE_H16 transfer
LDI	*+AR0(IR0),R0	
LSH	16,R0	
STI	R0,@HSC_PORT	
CALL	DSPDLAYLP	
CALL	NRDY_low_loop	;wait until HSC has processed command
LDI	LD_DOPF,R0	;Load DOPF command
LSH	16,R0	
STI	R0,@HSC_PORT	
CALL	DSPDLAYLP	
CALL	NRDY_low_loop	;wait until HSC has processed command
LDI	DOPF_L16,IR0	;DOPF_L16 transfer
LDI	*+AR0(IR0),R0	
LSH	16,R0	
STI	R0,@HSC_PORT	
CALL	DSPDLAYLP	
CALL	NRDY_low_loop	;wait until HSC has processed command
LDI	DOPF_H16,IR0	;DOPF_H16 transfer
LDI	*+AR0(IR0),R0	
LSH	16,R0	
STI	R0,@HSC_PORT	
CALL	DSPDLAYLP	
CALL	NRDY_low_loop	;wait until HSC has processed command
LDI	LD_FCSPACE,R0	;Load FCSPACE command
LSH	16,R0	
STI	R0,@HSC_PORT	
CALL	DSPDLAYLP	
CALL	NRDY_low_loop	;wait until HSC has processed command
LDI	FCSPACE_L16,IR0	;FCSPACE_L16 transfer
LDI	*+AR0(IR0),R0	
LSH	16,R0	
STI	R0,@HSC_PORT	
CALL	DSPDLAYLP	
CALL	NRDY_low_loop	;wait until HSC has processed command
LDI	FCSPACE_H16,IR0	;FCSPACE_H16 transfer
LDI	*+AR0(IR0),R0	
LSH	16,R0	
STI	R0,@HSC_PORT	
CALL	DSPDLAYLP	
CALL	NRDY_low_loop	;wait until HSC has processed command
LDI	LD_FLAGS,R0	;Load FLAGS command
LSH	16,R0	
STI	R0,@HSC_PORT	
CALL	DSPDLAYLP	
CALL	NRDY_low_loop	;wait until HSC has processed command
LDI	FLAGS_L16,IR0	;FLAGS_L16 transfer
LDI	*+AR0(IR0),R0	
LSH	16,R0	
STI	R0,@HSC_PORT	
CALL	DSPDLAYLP	
CALL	NRDY_low_loop	;wait until HSC has processed command
LDI	FLAGS_H16,IR0	;FLAGS_H16 transfer
LDI	*+AR0(IR0),R0	

```

LSH      16,R0
STI      R0,@HSC_PORT
CALL     DSPDLAYLP

CALL     NRDY_low_loop ;wait until HSC has processed command
LDI      LD_LOSCI,R0 ;Load LOSCI command
LSH      16,R0
STI      R0,@HSC_PORT
CALL     DSPDLAYLP

CALL     NRDY_low_loop ;wait until HSC has processed command
LDI      LOSCI_L16,IR0 ;LOSCI_L16 transfer
LDI      *+AR0(IR0),R0
LSH      16,R0
STI      R0,@HSC_PORT
CALL     DSPDLAYLP

CALL     NRDY_low_loop ;wait until HSC has processed command
LDI      LOSCI_H16,IR0 ;LOSCI_H16 transfer
LDI      *+AR0(IR0),R0
LSH      16,R0
STI      R0,@HSC_PORT
CALL     DSPDLAYLP

CALL     NRDY_low_loop ;wait until HSC has processed command
LDI      LD_LOCOM,R0 ;Load LOCOM command
LSH      16,R0
STI      R0,@HSC_PORT
CALL     DSPDLAYLP

CALL     NRDY_low_loop ;wait until HSC has processed command
LDI      LOCOM_L16,IR0 ;LOCOM_L16 transfer
LDI      *+AR0(IR0),R0
LSH      16,R0
STI      R0,@HSC_PORT
CALL     DSPDLAYLP

CALL     NRDY_low_loop ;wait until HSC has processed command
LDI      LOCOM_H16,IR0 ;LOCOM_H16 transfer
LDI      *+AR0(IR0),R0
LSH      16,R0
STI      R0,@HSC_PORT
CALL     DSPDLAYLP

CALL     NRDY_low_loop ;wait until HSC has processed command
LDI      LD_OFFSET,R0 ;Load OFFSET command
LSH      16,R0
STI      R0,@HSC_PORT
CALL     DSPDLAYLP

CALL     NRDY_low_loop ;wait until HSC has processed command
LDI      OFFSET_L16,IR0 ;OFFSET_L16 transfer
LDI      *+AR0(IR0),R0
LSH      16,R0
STI      R0,@HSC_PORT
CALL     DSPDLAYLP

CALL     NRDY_low_loop ;wait until HSC has processed command
LDI      OFFSET_H16,IR0 ;OFFSET_H16 transfer
LDI      *+AR0(IR0),R0
LSH      16,R0
STI      R0,@HSC_PORT
CALL     DSPDLAYLP

CALL     NRDY_low_loop ;wait until HSC has processed command
LDI      LD_TIMELO,R0 ;Load TIMELO command, autoclr up32 bits (TIMEHI)
LSH      16,R0
STI      R0,@HSC_PORT
CALL     DSPDLAYLP

CALL     NRDY_low_loop ;wait until HSC has processed command
LDI      TIMELO_L16,IR0 ;TIMELO_L16 transfer
LDI      *+AR0(IR0),R0
LSH      16,R0
STI      R0,@HSC_PORT
CALL     DSPDLAYLP

CALL     NRDY_low_loop ;wait until HSC has processed command
LDI      TIMELO_H16,IR0 ;TIMELO_H16 transfer
LDI      *+AR0(IR0),R0
LSH      16,R0
STI      R0,@HSC_PORT
CALL     DSPDLAYLP

POP      IR0
POP      AR0

```

```

                POPF        R0
                POP         R0
                POP         DP

                RETS                ;end of Ld_param_HSC subroutine
*****

                .globl        NRDY_low_loop

NRDY_low_loop:

                PUSH        DP
                PUSH        R0
                PUSHF        R0

NRDY_LOOP:      LDI          DSPLNPKG,DP

                LDI          @STATUS,R0
                LSH          -16,R0
                TSTB         NRDY_BIT,R0
                BNZ          NRDY_LOOP

                POPF        R0
                POP         R0
                POP         DP

                RETS                ;end of NRDY_low_loop subroutine
*****

                .globl        Rd_GTparam

Rd_GTparam:    PUSH        DP
                PUSH        ARO
                PUSH        IRO
                PUSH        R0
                PUSHF        R0

RD_GT_VARS:    LDI          DATAPG,DP
                LDI          @VAR_BASE,ARO
                LDI          DPMEMPG,DP

WAIT_GT_VAR:   LDI          @END_FILE,R0        ;check for end of data file
                CMPI        BITSET,R0
                BEQ          FINISHED
                LDI          @DAT_AVAIL,R0      ;check for data available
                CMPI        BITSET,R0
                BNE          WAIT_GT_VAR

READ_VAL:     LDI          BITCLR,R0
                STI          R0,@DAT_AVAIL
                LDI          @INDEX,IRO        ;read index in gt_vars array
                LDI          @DAT_VALUE,R0      ;read value
                STI          R0,*+ARO(IRO)      ;store value into gt_vars array
                LDI          BITSET,R0
                STI          R0,@DAT_ACK        ;send ack to PC
                B            WAIT_GT_VAR

FINISHED:     LDI          BITCLR,R0
                LDI          DPMEMPG,DP
                STI          R0,@END_FILE
                POPF        R0
                POP         R0
                POP         IRO
                POP         ARO
                POP         DP

                RETS                ;end of Rd_GTparam subroutine
*****

                .globl        Rd_HSCparam

Rd_HSCparam:   PUSH        DP
                PUSH        R0
                PUSHF        R0
                PUSH        ARO
                PUSH        IRO

RD_NEXT:      LDI          DPMEMPG,DP

WAIT_DAT:     LDI          @VAL_RDY,R0        ;check for next param ready
                CMPI        BITSET,R0

```

```

RD_DAT:
    BEQ      RD_DAT
    LDI      @END_ASCII,R0    ;also check for end of file
    CMPI     BITSET,R0
    BNE      WAIT_DAT
    B        WRAP_UP

    LDI      0,R0
    STI      R0,@VAL_RDY      ;clear VAL_RDY

    LDI      @VAL_IDX,IR0

    LDI      DPMEMPG,DP
    LDI      @VALUE,R0
    LDI      DATAPG,DP
    LDI      @HSC_addr,AR0
    LDI      BSSPG,DP
    STI      R0,*+AR0(IR0)    ;store next param in array

    LDI      BITSET,R0
    LDI      DPMEMPG,DP
    STI      R0,@VAL_ACK

    B        RD_NEXT

WRAP_UP:
    POP      IR0
    POP      AR0
    POPF     R0
    POP      R0
    POP      DP

    RETS                      ;end of Rd_HSCparam subroutine

```

\*\*\*\*\*

```

    .globl   SLOW_HOP

    .globl   hop_num
    .bss     hop_num,1
    .globl   hop_incr
    .bss     hop_incr,1

    .text

SLOW_HOP:
    PUSH     DP
    PUSH     R0
    PUSHF    R0
    PUSH     IOF

DO_AGAIN:
    LDI      0,R0
    LDI      BSSPG,DP
    STI      R0,@hop_num

GO_THRU_PN:
    LDI      BSSPG,DP
    LDI      @hop_num,R0
    AND      MASKL16,R0
    STI      R0,@LSB16
    LDI      @hop_num,R0
    LSH      -16,R0
    AND      MASKL16,R0
    STI      R0,@MSB16
    CALL     CHG_COM_DISP
    LDI      BSSPG,DP

    LDI      XF0_EN,IOF      ;enable XF0
    OR       XF_SET,IOF      ;put "1" on XF
    NOP
    NOP
    NOP
    AND      XF_CLR,IOF      ;put "0" on XF

HOLD:
    RPTS     @hop_dwell
    NOP

    OR       XF_SET,IOF      ;put "1" on XF
    NOP
    NOP
    AND      XF_CLR,IOF      ;put "0" on XF
    NOP
    NOP
    NOP
    NOP
    OR       XF_SET,IOF      ;put "1" on XF
    NOP

```

```

NOP
AND                XF_CLR, IOF        ;put "0" on XF

LDI                DPMEMPG, DP
LDI                @STOP_MOD3, R0
CMPI               1, R0
BEQ               EXIT_LOOP

LDI                BSSPG, DP
LDI                @hop_num, R0
ADDI              @hop_incr, R0
STI               R0, @hop_num
LDI                DATAPG, DP
CMPI               @HSC_endhop, R0
BLT               GO_THRU_FN
B                 DO_AGAIN

EXIT_LOOP:
LDI                DPMEMPG, DP
LDI                BITCLR, R0
STI               R0, @STOP_MOD3

POP               IOF
POPF              R0
POP               R0
POP               DP

RETS               ;end of SLOW_HOP subroutine
*****

.globl             START_NCO
START_NCO:
PUSH              DP
PUSH              R0
PUSHF             R0
PUSH              R1
PUSHF             R1
PUSH              R2
PUSHF             R2
PUSH              R3
PUSHF             R3

LDI                BSSPG, DP
LDI                @phs_rnded, R0
LDI                DATAPG, DP
AND               @D24_D31mask, R0
LSH               -8, R0
ADDI              @D24_D31addr, R0
LDI                BSSPG, DP
STI               R0, @Phase

LDI                @phs_rnded, R0
LDI                DATAPG, DP
AND               @D16_D23mask, R0
ADDI              @D16_D23addr, R0
LDI                BSSPG, DP
STI               R0, @Phase+1

LDI                @phs_rnded, R0
LDI                DATAPG, DP
AND               @D8_D15mask, R0
LSH               8, R0
ADDI              @D8_D15addr, R0
LDI                BSSPG, DP
STI               R0, @Phase+2

LDI                @phs_rnded, R0
LDI                DATAPG, DP
AND               @D0_D7mask, R0
LSH               16, R0
ADDI              @D0_D7addr, R0
LDI                BSSPG, DP
STI               R0, @Phase+3

CALL              COMMAND_CLK

POPF              R3
POP               R3
POPF              R2
POP               R2
POPF              R1
POP               R1
POPF              R0
POP               R0

```



POP DP

RETS ;end of START\_NCO subroutine

\*\*\*\*\*

.end

### B3. Coarse synchronization routine

```
*****
*               Program Name:  CSYNC.ASM               *
*               Author:       C. Tom                   *
*               Date edited:   30 March 1998           *
* Description:   Assembler code to be added to TMS_UL2.ASM which performs *
*               coarse synchronization. Based on state diagram approach *
*               which directs path based on current state and triggers.  *
*               *                                     *
*****

**** Subroutine declaration ****

                .globl      CRSE_SYNC

                .globl      CHG_STATE
                .globl      COMMAND_CLK
                .globl      DisableInt
                .globl      DSPDLAYLP
                .globl      INPUT_HYP_LOG
                .globl      NRDY_low_loop
                .globl      RETRV_HYP
                .globl      START_NCO

**** Miscellaneous constants ****

XF0_EN          .set        2h
XF_SET          .set        6h
XF_CLR          .set        0fffh

BITSET          .set        1
BITCLR          .set        0

STATE_ROW       .set        8
STATE_COL       .set        5

MASKL16         .set        0FFFFh

                .data
VAR_BASE        .word       gt_vars

* Indices for gt_vars parameters array

NUM_HOP         .set        0
MAX_FRM         .set        1
MAX_HOP         .set        2
PRB_START       .set        3
SRCH_LIM        .set        4
TIMES_4_CONFM   .set        5
RESP_BUF_SIZ    .set        6
MIN_DET_2_VER   .set        7
NUM_RETRANSMITS .set        8

* TRIGGERS for CSYNC routine

CMD_REC'D       .set        0           ;starting point of CSYNC routine
PRELIM_COMPL    .set        1           ;preliminary init complete
DET_REC'D       .set        2           ;detect received in synch resp
DET_CONFM       .set        3           ;detect has been confirmed
FALSE_DET       .set        4           ;false detect has occurred
PLINE_CLR       .set        5           ;pipeline of synch resp emptied
SRCH_RG_XCD     .set        6           ;search range for csync routine exceeded
PLINE_OFLO      .set        7           ;resp pipeline has overflowed

* STATES for CSYNC routine

IDLE            .set        0           ;starting state of CSYNC routine
PRELIM_INIT     .set        1           ;performing preliminary initialization
GEN_PROBES      .set        2           ;generating csync probes at appropriate time
VER_DETECT      .set        3           ;verifying a detect in the synch response
CLR_RESP_PIPE   .set        4           ;clearing the synch resp pipeline
GO_2_FSYNC      .set        5           ;detect confirmed, switching to fsync routine
SRCH_EXCEED     .set        6           ;search range exceeded procedure, send error msg
PLINE_ERR       .set        7           ;pipeline overflow during VER_DETECT, send err msg

                .data
STAT_ADDR       .word       LK_UP_BASE
HYP_LOG_ADDR    .word       hyp_log
HYP_FRM_ADDR    .word       hyp_frame
RESP_BUF_ADDR   .word       resp_buffer
BITS0_N_1       .word       3h          ;0000 0000 0000 0000 0000 0000 0000 0011
BITS2_31        .word       0FFFFFFFCh ;1111 1111 1111 1111 1111 1111 1111 1100
BITS2_N_3       .word       0Ch         ;0000 0000 0000 0000 0000 0000 0000 1100
```

\*\*\*\* STATE LOOKUP TABLE \*\*\*\*

```

.data                                ;STATE Look up table
; ----- trigger
; | --- curr_state
; | |
; \ \

LK_UP_BASE:
.word SYNC_INIT                     ;[0,0]=0
.word 0                             ;[0,1]=1
.word 0                             ;[0,2]=2
.word 0                             ;[0,3]=3
.word 0                             ;[0,4]=4
.word 0                             ;[1,0]=5
.word COMPUT_HYP                    ;[1,1]=6
.word 0                             ;[1,2]=7
.word 0                             ;[1,3]=8
.word 0                             ;[1,4]=9
.word 0                             ;[2,0]=10
.word 0                             ;[2,1]=11
.word CHK_DET                       ;[2,2]=12
.word 0                             ;[2,3]=13
.word CHK_DET                       ;[2,4]=14
.word 0                             ;[3,0]=15
.word 0                             ;[3,1]=16
.word 0                             ;[3,2]=17
.word RETURN                        ;[3,3]=18
.word 0                             ;[3,4]=19
.word 0                             ;[4,0]=20
.word 0                             ;[4,1]=21
.word 0                             ;[4,2]=22
.word CLR_PIPE                      ;[4,3]=23
.word 0                             ;[4,4]=24
.word 0                             ;[5,0]=25
.word 0                             ;[5,1]=26
.word 0                             ;[5,2]=27
.word 0                             ;[5,3]=28
.word COMPUT_HYP                    ;[5,4]=29
.word 0                             ;[6,0]=30
.word 0                             ;[6,1]=31
.word OUT_A_RANGE                   ;[6,2]=32
.word 0                             ;[6,3]=33
.word 0                             ;[6,4]=34
.word 0                             ;[7,0]=35
.word 0                             ;[7,1]=36
.word 0                             ;[7,2]=37
.word OFLO_RESP                     ;[7,3]=38
.word 0                             ;[7,4]=39

```

\*\*\*\* Data page pointers \*\*\*\*

```

INIPG      .set      0
DPMEMPG    .set      3h
BSSPG      .set      80h
DATAPG     .set      0h
DSPLNKPG   .set      80h
BUSPG      .set      80h
DEBUGPG    .set      80h

```

\*\*\*\*\* DPMEM addresses \*\*\*\*

```

DPBASE     .set      30000h

```

\* DPBASE to DPBASE+18h used in TMS\_UL2.asm

```

HYP_AVAIL  .set      DPBASE+19h
USER_HYP   .set      DPBASE+1Ah
HYP_ACK    .set      DPBASE+1Bh
DEBUG_AVAIL .set      DPBASE+1Ch
DEBUG_VALUE .set      DPBASE+1Dh
DEBUG_ACK  .set      DPBASE+1Eh
CHK_AVAIL  .set      DPBASE+1Fh
CHK_VALUE  .set      DPBASE+20h
CHK_ACK    .set      DPBASE+21h
FRM_REF    .set      DPBASE+22h
CQ_RESP    .set      DPBASE+23h
C1_RESP    .set      DPBASE+24h
F0_RESP    .set      DPBASE+25h
F1_RESP    .set      DPBASE+26h
RESP_AVAIL .set      DPBASE+27h
RDY_4_RESP .set      DPBASE+28h
RESP_ACK   .set      DPBASE+29h
RNG_XCDED  .set      DPBASE+2Ah
CSYNC_OK   .set      DPBASE+2Bh
LOG_END    .set      DPBASE+2Eh
PLINE_FLAG .set      DPBASE+2Fh

```

```

TOO_MANY_HYPS    .set          DPBASE+35h
FRM_NOT_FOUND    .set          DPBASE+36h

```

\* DPBASE+30h to DPBASE+34h used in TMS\_UL2.asm

\*\*\*\* GT i/f board addresses \*\*\*\*

```

COMMAND          .set          800004h          ;WRITE only
STATUS           .set          800004h          ;READ only
NCO_CMD          .set          800005h          ;WRITE only
INTRPT_PORT      .set          800005h          ;READ only
HSC_PORT         .set          800006h          ;WRITE only
FSK_FRM          .set          800007h          ;WRITE only

```

\*\*\*\* HSC commands and miscellaneous \*\*\*\*

```

HSC_endhop       .data          .word          0ffffffh          ;16777215 (max hop number for HSC)

SYNC_BIT         .set          2                ;SYNC on bit D1 of GT status
NRDY_BIT         .set          4                ;bit D2 on GT status (DSPLINK)

STOP_HSC         .set          0
RUN_HSC          .set          8000h
CHG_IMMED        .set          0501h
CHG_HOP          .set          0500h
LD_LATCH         .set          0300h
LD_BASE          .set          030Ch
LD_BWSCALE       .set          0306h
LD_DOPF          .set          0310h
LD_FCSPACE       .set          030Ah
LD_FLAGS         .set          031Ah
LD_FSKCHAN       .set          0308h
LD_HOP           .set          0304h
LD_LOSCI         .set          0318h
LD_LOCOM         .set          0312h
LD_OFFSET        .set          030Eh
LD_TIMELO        .set          0314h
LD_TIMEHI        .set          0316h

ULSYNC_CMD       .set          500Bh          ;GO TO ULSYNC MODE, channel 1, bin 3
ULGO_BASE        .set          200h          ;BASE VALUE OF ULGO COMMAND FOR CSYNC

```

\*\*\*\* NCO constants \*\*\*\*

```

NCO_DLAY         .data          .set          13A14CFh          ;resulting phase incr to delay NCO clock
                                     ; by 0.5 hop over 320 hops

```

\*\*\*\* Reserve section in RAM block 0 for debug log \*\*\*\*

```

DEBUG_ADDR       .data          .word          809A00h

```

\*\*\*\* Reserve memory in .bss for variables \*\*\*\*

```

.globl          curr_state
.bss            curr_state,1
.globl          trigger
.bss            trigger,1
.globl          hyp_used
.bss            hyp_used,1
.globl          hyp_offset
.bss            hyp_offset,1
.globl          nxt_prb_frm
.bss            nxt_prb_frm,1
.globl          act_prb_frm
.bss            act_prb_frm,1
.globl          act_prb_hop
.bss            act_prb_hop,1
.globl          start_timef
.bss            start_timef,1
.globl          start_timeh
.bss            start_timeh,1
.globl          assigned_f
.bss            assigned_f,1
.globl          tms_csync_rdy
.bss            tms_csync_rdy,1
.globl          brst0_flg
.bss            brst0_flg,1
.globl          brst1_flg
.bss            brst1_flg,1
.globl          user_hyp_off
.bss            user_hyp_off,1
.globl          iter_hyp
.bss            iter_hyp,1

```

```

.globl      first_prb
.bss        first_prb,1
.globl      prb_cmd
.bss        prb_cmd,1
.globl      coarse0
.bss        coarse0,1
.globl      coarse1
.bss        coarse1,1
.globl      ref_frame
.bss        ref_frame,1
.globl      hyp_log
.bss        hyp_log,128
.globl      hyp_frame
.bss        hyp_frame,128
.globl      hyp_index
.bss        hyp_index,1
.globl      resp_buffer
.bss        resp_buffer,25
.globl      buff_idx
.bss        buff_idx,1
.globl      ver_count
.bss        ver_count,1
.globl      new_state
.bss        new_state,1
.globl      valid_resp_flg
.bss        valid_resp_flg,1
.globl      last_resp_clrd
.bss        last_resp_clrd,1
.globl      last_resp_in
.bss        last_resp_in,1
.globl      last_hyp_tested
.bss        last_hyp_tested,1
.globl      pipe_idx
.bss        pipe_idx,1
.globl      dbug_idx
.bss        dbug_idx,1
.globl      dbug_adr
.bss        dbug_adr,1
.globl      dbug_cnt
.bss        dbug_cnt,1
.globl      re_tx_cnt
.bss        re_tx_cnt,1
.globl      det_count
.bss        det_count,1
.globl      ND_count
.bss        ND_count,1
.globl      DD_count
.bss        DD_count,1
.globl      DN_count
.bss        DN_count,1
.globl      hop_ref
.bss        hop_ref,1
.globl      frm_ref
.bss        frm_ref,1

**** Variables defined elsewhere ****

.globl      hop_cnt
.globl      frm_cnt
.globl      gt_vars
.globl      original_phase
.globl      phs_rnded

*****

**** Program begins here ****

*****

.text

CRSE_SYNC:

*****
; STATE 1: PRELIM_INIT
*****

SYNC_INIT:

        LDI        BSSPG,DP        ;update current state
        LDI        PRELIM_INIT,R0
        STI        R0,@curr_state

        LDI        DATAPG,DP
        LDI        @VAR_BASE,AR2    ;set up AR2

;Step 1: Compute precalc time required by HSC

```

```

        LDI        BSSPG,DP        ;allow time for HSC to precompute
        LDI        @frm_cnt,R0     ; hop frequencies for CSYNC
        ADDI       3,R0            ;start @ 2nd frame after current + 6% = 3rd frm
        CMPI       **AR2(MAX_FRM),R0;check if number exceeds max frame #
        BLE        UNDER_LIM

OVER_LIM:
        LDI        **AR2(MAX_FRM),R1;rollover frame
        ADDI       1,R1
        SUBI       R1,R0

UNDER_LIM:
        STI        R0,@start_timef ;used to find first valid synch resp
        STI        R0,@nxt_prb_frm
        LDI        DATAPG,DP
        TSTB       @BITS0_N_1,R0   ;round to next multiple of 4
        BZ         SET_TIME        ;already a multiple of 4

NXT_MULT4:
        LDI        DATAPG,DP
        AND        @BITS2_31,R0    ;mask off 2 LSBs
        ADDI       4,R0
        CMPI       **AR2(MAX_FRM),R0;check if number exceeds max frame #
        BLE        LT_LIM

GT_LIM:
        LDI        **AR2(MAX_FRM),R1;rollover frame
        ADDI       1,R1
        SUBI       R1,R0

LT_LIM:
        LDI        BSSPG,DP
        STI        R0,@start_timef
        STI        R0,@nxt_prb_frm

;Step 2: set TIMELO and TIMEHI on HSC due to precalc time

SET_TIME:
        MPYI       **AR2(NUM_HOP),R0;change start_time in frames to hops
        LDI        BSSPG,DP        ;MAKE SURE NUM_HOP IS 320!!!
        STI        R0,@start_timeh

        LDI        DSPLNKPG,DP     ;send LD_TIMELO command
        CALL       NRDY_low_loop
        LDI        LD_TIMELO,R0
        LSH        16,R0
        STI        R0,@HSC_PORT
        CALL       DSPDLAYLP

        CALL       NRDY_low_loop
        LDI        BSSPG,DP
        LDI        @start_timeh,R0 ;send 16 LSBs of start_time
        AND        MASKL16,R0
        LSH        16,R0
        LDI        DSPLNKPG,DP
        STI        R0,@HSC_PORT
        CALL       DSPDLAYLP

        CALL       NRDY_low_loop
        LDI        BSSPG,DP
        LDI        @start_timeh,R0 ;send 16 MSBs of start_time
        LDI        DSPLNKPG,DP
        STI        R0,@HSC_PORT
        CALL       DSPDLAYLP

;Step 2a: Initialize variables

        LDI        BSSPG,DP
        LDI        BITCLR,R0
        STI        R0,@hyp_used    ;hyp_used = 0
        STI        R0,@assigned_f ;assigned_f = 0
        STI        R0,@brst0_flg  ;brst0_flg = 0
        STI        R0,@brst1_flg  ;brst1_flg = 0
        STI        R0,@hyp_offset ;hyp_offset = 0
        STI        R0,@tms_csync_rdy;tms_csync_rdy = 0
        STI        R0,@valid_resp_flg ;valid_resp_flg = 0
        STI        R0,@new_state  ;new_state = 0
        STI        R0,@buff_idx   ;buff_idx = 0
        STI        R0,@last_resp_clrd ;last_resp_clrd = 0
        STI        R0,@last_resp_in ;last_resp_in = 0
        STI        R0,@ver_count  ;ver_count = 0
        STI        R0,@last_hyp_tested ;last_hyp_tested = 0
        STI        R0,@pipe_idx   ;pipe_idx = 0
        STI        R0,@hyp_index  ;hyp_index = 0

        LDI        1,R1            ;init hyp_log and hyp_frame to 9999
        LDI        R1,IR1
        LDI        9999,R0
        LDI        DATAPG,DP
        LDI        @HYP_LOG_ADDR,AR0
        LDI        @HYP_FRM_ADDR,AR1

```

```

                LDI            127,RC            ;set repeat counter
                RPTB
                STI            R0,*ARO++(IR1)
ZRO_LOOP:      STI            R0,*AR1++(IR1)      ;store '9999' in hyp_log and hyp_frame

                LDI            BSSPG,DP

                LDI            ULGO_BASE,R0
                STI            R0,@prb_cmd

WAIT_4_HYP:
                LDI            DPMEMPG,DP        ;in case user wants to start at
                LDI            @HYP_AVAIL,R0      ;other than PRB_START=288
                CMPI           BITSET,R0
                BNE            WAIT_4_HYP

GET_HYP:
                LDI            BITCLR,R0
                STI            R0,@HYP_AVAIL
                LDI            @USER_HYP,R0
                LDI            BSSPG,DP
                STI            R0,@user_hyp_off ;read user hypothesis offset
                LDI            BITSET,R0
                LDI            DPMEMPG,DP
                STI            R0,@HYP_ACK

                LDI            BSSPG,DP
                LDI            @start_timef,R0
                STI            R0,@act_prb_frm    ;act_prb_frm = start_timef

                CALL           INPUT_HYP_LOG      ;store 1st hyp_offset (default 0) in hyp_log

                LDI            BSSPG,DP
                LDI            *+AR2(PRB_START),R0 ;PRB_START = default hop # to start synch probes
                LDI            @user_hyp_off,R1
                ADDI           R1,R0
                STI            R0,@first_prb     ;starting point for binary search
                STI            R0,@act_prb_hop    ;act_prb_hop = PRB_START+user_hyp_off

* Final check to see if adjustment of frame and hop number needed
* because of user hyp offset

                CMPI           *+AR2(NUM_HOP),R0 ;check if number exceeds max hop #
                BLT            TX_CSINC_CMD      ;will branch on the first iteration/run
                SUBI           *+AR2(NUM_HOP),R0 ;rollover hop
                STI            R0,@act_prb_hop
                LDI            @nxt_prb_frm,R0
                ADDI           1,R0
                CMPI           *+AR2(MAX_FRM),R0 ;check if number exceeds max frm #
                BLE            BELOW_LIM
ABOV_LIM:
                LDI            *+AR2(MAX_FRM),R1 ;rollover frame
                ADDI           1,R1
                SUBI           R1,R0
BELOW_LIM:
                STI            R0,@act_prb_frm

; at this point, HSC has been set up for coarse synch, switch to
; ULSYNC mode for HSC and check for SYNC line from status before
; proceeding.

;Step 3: Change HSC to ULSYNC mode

TX_CSINC_CMD:
                CALL           NRDY_low_loop
                LDI            ULSYNC_CMD,R0
                LSH            16,R0
                LDI            DSPLNKPG,DP
                STI            R0,@HSC_PORT      ;send ULSYNC_CMD to HSC
                CALL           DSPDLAYLP

WAIT_4_SYNC:
                LDI            DSPLNKPG,DP
                LDI            @STATUS,R0
                LSH            -16,R0
                CALL           DSPDLAYLP
                TSTB           SYNC_BIT,R0
                BZ             WAIT_4_SYNC

;Step 4: now, allow precalc time for HSC to finish calculations

PRECALC_WAIT:
                LDI            BSSPG,DP
                LDI            @frm_cnt,R0        ;wait until frm_cnt = (start_timef - 1)
                LDI            @start_timef,R1
                SUBI           1,R1

```

```

                                CMPI            R0,R1
                                BNZ            PRECALC_WAIT
PAST_HOP0:
                                LDI            @hop_cnt,R0
                                CMPI            0,R0
                                BEQ            PAST_HOP0

;//////////////////////////////////////
; Trigger = PRELIM_COMPL, GO TO STATE 2 (GEN_PROBES)
;//////////////////////////////////////

;*****
; STATE 2: GEN_PROBES
;*****

COMPUT_HYP:

                                LDI            BSSPG,DP            ;update current state
                                LDI            GEN_PROBES,R0
                                STI            R0,@curr_state

                                LDI            BSSPG,DP            ;set flag for ISR
                                LDI            BITSET,R0
                                STI            R0,@tms_csync_rdy

                                LDI            BSSPG,DP
                                LDI            @hyp_used,R0        ;check if hyp transmitted
                                CMPI            BITSET,R0
                                BNE            CHK_FRM            ;will branch on first iter/run

                                LDI            BITCLR,R0
                                LDI            BSSPG,DP
                                STI            R0,@hyp_used        ;clear flag for hypothesis used
                                STI            R0,@assigned_f      ;clear flag for ISR
                                STI            R0,@brst0_flg      ;clear flag for brst0 probes (ISR)
                                STI            R0,@brst1_flg      ;clear flag for brst1 probes (ISR)
                                LDI            ULGO_BASE,R0        ;reset prb_cmd
                                STI            R0,@prb_cmd

                                B            CONT_SRCH            ;TEST ONLY

;                                LDI            BSSPG,DP            ;check if search range is exceeded
;                                ABSI            @hyp_offset,R0
;                                CMPI            *+AR2(SRCH_LIM),R0
;                                BLT            CONT_SRCH

;//////////////////////////////////////
; Trigger = SRCH_RG_XCD, GO TO STATE 6 (SRCH_EXCEED)
;//////////////////////////////////////

                                LDI            BSSPG,DP
                                LDI            SRCH_RG_XCD,R0
                                STI            R0,@trigger
                                CALL            CHG_STATE
                                LDI            BSSPG,DP
                                LDI            @new_state,R2
                                B            R2

;//////////////////////////////////////

CONT_SRCH:
                                LDI            @nxt_prb_frm,R0      ;comp nxt frm to tx csync
                                ADDI            4,R0                ; probes
                                CMPI            *+AR2(MAX_FRM),R0 ;check if frame # exceeded
                                BLE            KEEP_FRM
                                LDI            *+AR2(MAX_FRM),R1 ;rollover frame
                                ADDI            1,R1
                                SUBI            R1,R0

KEEP_FRM:
                                STI            R0,@nxt_prb_frm
                                STI            R0,@act_prb_frm

NEXT_HYP:
                                LDI            BSSPG,DP
                                LDI            @hyp_offset,R0
                                CMPI            0,R0
                                BLE            NXT_HYP_POS

NXT_HYP_NEG:
                                NEGI            R0,R1            ;negate hypothesis offset
                                LDI            BSSPG,DP
                                STI            R1,@hyp_offset

                                CALL            INPUT_HYP_LOG      ;store hyp_offset in HYP_LOG & HYP_FRAME #

                                LDI            BSSPG,DP
                                LDI            @first_prb,R2      ;compute actual hop # to tx
                                ADDI            R1,R2            ; csync probe
                                STI            R2,@act_prb_hop
                                B            CHK_FRM

```



```

NXT_HYP_POS:
    NEGI        R0,R1        ;negate hypothesis offset
    ADDI        1,R1        ;incr hypothesis
    LDI        BSSPG,DP
    STI        R1,@hyp_offset

    CALL        INPUT_HYP_LOG    ;store hyp_offset in hyp_log and hyp_frame

    LDI        BSSPG,DP
    LDI        @first_prb,R2    ;compute actual hop # to tx
    ADDI        R1,R2        ; csync probe, adj frm # if
    STI        R2,@act_prb_hop    ; necessary
    CMPI        *+AR2(NUM_HOP),R2;check if number exceeds max hop #
    BLT        CHK_FRM
    SUBI        *+AR2(NUM_HOP),R2;rollover hop
    STI        R2,@act_prb_hop
    LDI        @nxt_prb_frm,R2
    ADDI        1,R2
    CMPI        *+AR2(MAX_FRM),R2;check if number exceeds max frm #
    BLE        VALID_FRM
    LDI        *+AR2(MAX_FRM),R1;rollover frame
    ADDI        1,R1
    SUBI        R1,R2

VALID_FRM:
    STI        R2,@act_prb_frm

;up to here act_prb_hop & act_prb_frm have been calculated
; Now, go into loop of generating probes and collecting responses

CHK_FRM:
    LDI        BSSPG,DP
    LDI        @assigned_f,R0    ;check if already in assigned frame
    CMPI        BITSET,R0
    BEQ        CHK_RET_LNK

    LDI        @act_prb_frm,R0
    LDI        @frm_cnt,R1
    CMPI        R0,R1        ;check if current is assigned frame
    BNE        CHK_RET_LNK

SET_TXPRB:
    ;here, arrived at assigned frame

    LDI        BITSET,R0        ;set flag to signal assigned frame
    STI        R0,@assigned_f

CHK_RET_LNK:

    LDI        DPMPG,DP
    LDI        @RESP_AVAIL,R0
    CMPI        BITSET,R0
    BNE        COMPUT_HYP        ;go back to top of loop

RD_RESP:
    LDI        DPMPG,DP
    LDI        BITCLR,R0        ;clear flag
    STI        R0,@RESP_AVAIL

    LDI        @FRM_REF,R0        ;store synch response
    LDI        @C0_RESP,R1
    LDI        @C1_RESP,R2
    LDI        BSSPG,DP
    STI        R0,@ref_frame
    STI        R1,@coarse0        ;coarse synch response for ch0
    STI        R2,@coarse1        ;coarse synch response for ch1

    LDI        BITSET,R0        ;ack synch response
    LDI        DPMPG,DP
    STI        R0,@RESP_ACK

; Check if response frame # is a multiple of 4 (probe every 4 frame - user allocation assumed.)
; Check also if frame # is later than the start_timef -> set flag

    LDI        BSSPG,DP
    LDI        @ref_frame,R0
    LDI        DATAPG,DP
    TSTB        @BITS0_N_1,R0
    BNZ        RESP_WRAP_UP        ;not a mult of 4, disregard response

;
;    LDI        BSSPG,DP        ;check if past starting point of probe frames
;    LDI        @valid_resp_flg,R3
;    CMPI        BITSET,R3
;    BEQ        TEST_RESP

;
;    LDI        BSSPG,DP
;    LDI        @ref_frame,R0

```

```

;          LDI          @start_timef,R3
;          CMPI         R0,R3          ;R3 - R0 = start_timef - ref_frame
;          BGT          RESP_WRAP_UP   ;invalid response, disregard

;          LDI          BITSET,R3
;          STI          R3,@valid_resp_flg

; Current user allocated chl for coarse synch probes.
; See ULSYNC_CMD description above

TEST_RESP:
          CMPI          0,R2          ;R2 <- coarse1
          BEQ           RESP_WRAP_UP   ;no "DETECT" for either burst

;//////////////////////////////////////
; Trigger = DET_RECD, SAVE CURRENT HYP/PROBE SETTINGS, GO TO STATE 3 (VER_DETECT)
;//////////////////////////////////////

          LDI          BSSPG,DP        ;save current hypothesis for resume probing
          LDI          @hyp_offset,R0
          STI          R0,@last_hyp_tested

          LDI          ULGO_BASE,R0     ;reset prb_cmd
          STI          R0,@prb_cmd

          LDI          BITCLR,R0        ;clear variables related to tx of probes
          STI          R0,@hyp_used
          STI          R0,@assigned_f
          STI          R0,@brst0_flg
          STI          R0,@brst1_flg
          LDI          BSSPG,DP

          LDI          DET_RECD,R0      ;set trigger
          STI          R0,@trigger
          CALL          CHG_STATE
          LDI          BSSPG,DP
          LDI          @new_state,R2
          B            R2              ;go to STATE 3
;//////////////////////////////////////

RESP_WRAP_UP:
          B            COMPUT_HYP

;*****
; STATE 3: VER_DETECT
;*****

CHK_DET:
          LDI          BSSPG,DP        ;modify current state
          LDI          VER_DETECT,R0
          STI          R0,@curr_state

          LDI          @last_resp_in,R0 ;init buffer index for response pipeline
          STI          R0,@buff_idx

; initialize variables for verifying detect

          LDI          BSSPG,DP
          LDI          BITCLR,R0
          STI          R0,@re_tx_cnt
          STI          R0,@det_count
          STI          R0,@ND_count
          STI          R0,@DD_count
          STI          R0,@DN_count

          CALL          RETRV_HYP        ;retrieve hypothesis to verify

          LDI          BSSPG,DP
          LDI          @nxt_prb_frm,R0  ;compute next frame for probe
          ADDI          4,R0
          CMPI          @frm_cnt,R0     ;check if nxt_prb_frm is already passed
          BGT          NXT_2_PRB
          ADDI          4,R0            ;go to next multiple of 4

NXT_2_PRB:
          CMPI          *+AR2(MAX_FRM),R0;check if number exceeds max frm #
          BLE          FRM_NO_CHG
          LDI          *+AR2(MAX_FRM),R1
          ADDI          1,R1
          SUBI          R1,R0

FRM_NO_CHG:
          STI          R0,@nxt_prb_frm
          STI          R0,@act_prb_frm
          STI          R0,@start_timef ;used to find when synch resp are valid

          LDI          @hyp_offset,R0   ;compute act_prb_hop with offset

```

```

        LDI            @first_prb,R1      ;first_prb includes user offset @ beginning
        ADDI           R0,R1
        STI            R1,@act_prb_hop
        CMPI           *+AR2(NUM_HOP),R1;check if number exceeds max hop #
        BLT            REPEAT_CHK
        SUBI           *+AR2(NUM_HOP),R1;max hop# exceeded, roll over
        STI            R1,@act_prb_hop
        LDI            @nxt_prb_frm,R1    ;adjust frm# because of roll over
        ADDI           1,R1
        CMPI           *+AR2(MAX_FRM),R1;check if number exceeds max frm #
        BLE            FRM_OK
        LDI            *+AR2(MAX_FRM),R2
        ADDI           1,R2
        SUBI           R2,R1
FRM_OK:
        STI            R1,@act_prb_frm

; At this point, act_prb_hop and act_prb_frm have been calculated for VER_DETECT

REPEAT_CHK:
        LDI            BSSPG,DP
        LDI            @hyp_used,R0
        CMPI           BITSET,R0
        BNE            WAIT_4_FRM

        LDI            BSSPG,DP
        LDI            BITCLR,R0
        STI            R0,@hyp_used
        STI            R0,@assigned_f
        STI            R0,@brst0_flg
        STI            R0,@brst1_flg
        LDI            ULGO_BASE,R0
        STI            R0,@prb_cmd

        LDI            BSSPG,DP          ;update nxt_prb_frm
        LDI            @nxt_prb_frm,R0
        ADDI           4,R0
        CMPI           *+AR2(MAX_FRM),R0;check if number exceeds max frm #
        BLE            STORE_FRM
        LDI            *+AR2(MAX_FRM),R1;rollover frame
        ADDI           1,R1
        SUBI           R1,R0
STORE_FRM:
        STI            R0,@nxt_prb_frm

        LDI            @act_prb_frm,R0   ;compute next probe frame
        ADDI           4,R0               ; using act_prb_frm to avoid recal
        CMPI           *+AR2(MAX_FRM),R0; act_prb_hop should be the same
        BLE            FRM_UNDER         ;done separately in case of straddling
                                         ; of frame boundaries betw nxt_prb_frm

FRM_OVER:
        LDI            *+AR2(MAX_FRM),R1;rollover frame
        ADDI           1,R1
        SUBI           R1,R0             ; and act_prb_frm
FRM_UNDER:
        STI            R0,@act_prb_frm

WAIT_4_FRM:
        LDI            BSSPG,DP
        LDI            @assigned_f,R0
        CMPI           BITSET,R0
        BEQ            LOOK_4_RESP

        LDI            @act_prb_frm,R0
        LDI            @frm_cnt,R1
        CMPI           R0,R1
        BNE            LOOK_4_RESP

GO_PROBE:
        LDI            BITSET,R0
        STI            R0,@assigned_f

LOOK_4_RESP:
        LDI            DPMEMPG,DP
        LDI            @RESP_AVAIL,R0
        CMPI           BITSET,R0
        BNE            REPEAT_CHK

GET_RESP:
        LDI            BITCLR,R0        ;clear flag
        STI            R0,@RESP_AVAIL
        LDI            @FRM_REF,R0      ;obtain responses
        LDI            @CO_RESP,R1
        LDI            @C1_RESP,R2
        LDI            BSSPG,DP
        STI            R0,@ref_frame
        STI            R1,@coarse0

```

```

        STI            R2,@coarse1

        LDI            BITSET,R3            ;ACK resp
        LDI            DPMEMPG,DP
        STI            R3,@RESP_ACK

; Check response is a mult of 4 and references frame later than start_timef

        LDI            BSSPG,DP
        LDI            @ref_frame,R0
        LDI            DATAPG,DP
        TSTB           @BITS0_N_1,R0      ;look at lower 2 bits to see if a mult of 4
        BNZ            CLOSE_RESP         ;not a mult of 4, disregard response

;::::;
; FOR TEST ONLY
;::::;
        B              LOOK_4_DET         ;test - go directly to testing response
;::::;

;
        LDI            BSSPG,DP            ;check if frame # > start_timef
        LDI            @ref_frame,R0      ; for valid response
;
        LDI            @start_timef,R1
;
        CMPI           R0,R1              ;R1 - R0 = start_timef - ref_frame
;
        BLE            LOOK_4_DET         ;valid verify response
;                                       ; Otherwise,

SAV_2_PIPE:
        LDI            BSSPG,DP            ;check to see if anymore room in response
        LDI            @buff_idx,R0       ; pipeline buffer
        CMPI           @last_resp_clrd,R0
        BNE            ASSMBL_RESP

;//////////////////////////////////////////
; Trigger = PLINE_OFLO, go to state 7 (OFLO_RESP)
;//////////////////////////////////////////

        LDI            BSSPG,DP
        LDI            PLINE_OFLO,R0
        STI            R0,@trigger
        CALL           CHG_STATE
        LDI            BSSPG,DP
        LDI            @new_state,R2
        B              R2
;//////////////////////////////////////////

ASSMBL_RESP:
        LDI            @ref_frame,R0      ;save responses arriving after "detect"
        LDI            @coarse0,R1        ; frame # in 16MSB
        LDI            @coarse1,R2        ; ch0 coarse in D0-D1
        LSH            16,R0              ; ch1 coarse in D2-D3
        LSH            2,R2
        ADDI           R2,R0              ;append ch1 coarse resp
        ADDI           R1,R0              ;append ch0 coarse resp
        LDI            DATAPG,DP
        LDI            @RESP_BUF_ADDR,AR0
        LDI            BSSPG,DP
        LDI            @buff_idx,R3
        LDI            R3,IR0
        STI            R0,*+AR0(IR0)
        ADDI           1,R3              ;increment buffer index
        CMPI           *+AR2(RESP_BUF_SIZ),R3 ;check for rollover of buffer index
        BLT            SAV_BUF_IDX
        SUBI           *+AR2(RESP_BUF_SIZ),R3

SAV_BUF_IDX:
        STI            R3,@buff_idx
        B              CLOSE_RESP

; See if "detect" received again
; Again, ch1 for current user assumed

LOOK_4_DET:

        LDI            BSSPG,DP            ;increment retransmit count
        LDI            @re_tx_cnt,R0
        ADDI           1,R0
        STI            R0,@re_tx_cnt

        LDI            BSSPG,DP            ;check if "detect" received
        LDI            @coarse1,R2
        CMPI           0,R2
        BNE            INCR_TALLY         ; "detect" recd again, incr counter

;no detect received, so check if # times retransmit completed

        B              CHK_NUM_RETX

INCR_TALLY:
        LDI            BSSPG,DP            ;increase # of "detect"s count

```

```

        LDI        @det_count,R0
        ADDI       1,R0
        STI        R0,@det_count

        CMPI       1,R2          ;R2 still has coarse1
        BNE        NOT_ND        ;determine whether "ND", "DD", or "DN" rec'd
        LDI        @ND_count,R0  ;"ND" received, increment ND_count
        ADDI       1,R0
        STI        R0,@ND_count
        B          CHK_NUM_DET    ;check if enough "detect"s for confm
NOT_ND:
        CMPI       2,R2          ;check for "DN" received
        BNE        ITS_DD
        LDI        @DN_count,R0  ;"DN" received, increment DN_count
        ADDI       1,R0
        STI        R0,@DN_count
        B          CHK_NUM_DET    ;check if enough "detect"s for confm
ITS_DD:
        LDI        @DD_count,R0
        ADDI       1,R0
        STI        R0,@DD_count
CHK_NUM_DET:
        LDI        BSSPG,DP      ;check if minimum # of detects received for confm
        LDI        @det_count,R0
        CMPI       *+AR2(MIN_DET_2_VER),R0
        BLT        CHK_NUM_RETX  ;if not enough yet, check if # retransmit completed
                                   ;otherwise,

;////////////////////////////////////
; Trigger = DET_CONFM, GO TO STATE 4 (GO_2_FSYNC) (for now, RETURN)
;////////////////////////////////////

;FOR TEST, GO STRAIGHT TO CHANGE:

        B          CHANGE        ;FOR TEST ONLY

; first compare ND_count with both DD_ and DN_count. if ND > DD and DN, then
; have to delay NCO by 180 (see synch response scenarios)

        LDI        BSSPG,DP
        LDI        @ND_count,R0
        CMPI       @DD_count,R0  ;compare ND and DD
        BLT        CHANGE        ;if ND < DD, then no need to adjust clock
        CMPI       @DN_count,R0  ;compare ND and DN
        BLT        CHANGE        ;if ND < DN, then no need to adjust clock
DLAY_BY_180:
                                   ;ND > both DD and DN, therefore, delay clock
        LDI        DATAPG,DP
        LDI        @NCO_DELAY,R0
        LDI        BSSPG,DP
        STI        R0,@phs_rnded

        CALL       START_NCO      ;change to slower frequency

        LDI        BSSPG,DP
        LDI        @hop_cnt,R0
        LDI        @frm_cnt,R1
        STI        R0,@hop_ref    ;save current hop and frame #
        STI        R0,@frm_ref

WAIT_320_HOPS:
        LDI        BSSPG,DP
        LDI        @hop_cnt,R0
        CMPI       @hop_ref,R0    ;compare hop # first, then frm #
        BNE        WAIT_320_HOPS ;hop # still not cycled through, keep waiting
        LDI        @frm_cnt,R0
        CMPI       @frm_ref,R0
        BEQ        WAIT_320_HOPS  ;frm # has not advanced (<320hops), keep waiting

        LDI        BSSPG,DP
        LDI        @original_phase,R0
        STI        R0,@phs_rnded

        CALL       START_NCO      ;change back to original NCO frequency

CHANGE:
        LDI        BSSPG,DP      ;set trigger
        LDI        DET_CONFM,R0
        STI        R0,@trigger
        CALL       CHG_STATE
        LDI        BSSPG,DP
        LDI        @new_state,R2
        B          R2            ;go to new state (state 4)

;check if hypothesis has been retransmitted enough times to determine whether to continue

```

```

CHK_NUM_RETX:
    LDI        BSSPG,DP
    LDI        @re_tx_cnt,R0
    CMPI       *+AR2(NUM_RETRANSMITS),R0
    BLT        CLOSE_RESP

;otherwise, finish all retransmits and not enough "detects" received to confm
;/////////////////////////////////////////////////////////////////
; Trigger = FALSE_DET, GO TO STATE 5 (CLR_RESP_PIPE)
;/////////////////////////////////////////////////////////////////

    LDI        BSSPG,DP            ;save loc of last entry into resp_buff
    LDI        @buff_idx,R0
    STI        R0,@last_resp_in

    LDI        ULGO_BASE,R0        ;reset prb_cmd
    STI        R0,@prb_cmd

    LDI        BITCLR,R0           ;clear variables related to tx of probes
    STI        R0,@hyp_used
    STI        R0,@assigned_f
    STI        R0,@brst0_flg
    STI        R0,@brst1_flg

    LDI        BSSPG,DP           ;set trigger
    LDI        FALSE_DET,R0
    STI        R0,@trigger
    CALL       CHG_STATE
    LDI        BSSPG,DP
    LDI        @new_state,R2
    B          R2                 ;go to new state (state 5)
;/////////////////////////////////////////////////////////////////

CLOSE_RESP:
    B          REPEAT_CHK

*****
; State 4: GO_2_FSYNC (*temporarily set to RETURN*)
*****

RETURN:
    LDI        BSSPG,DP
    LDI        GO_2_FSYNC,R0      ;modify current state
    STI        R0,@curr_state

    LDI        BITCLR,R0         ;clear flags for isr
    STI        R0,@hyp_used
    STI        R0,@assigned_f
    STI        R0,@brst0_flg
    STI        R0,@brst1_flg

    LDI        @hop_cnt,R0        ;adjust hop and frame counter
    ADDI       @hyp_offset,R0
    STI        R0,@hop_cnt
    CMPI       *+AR2(NUM_HOP),R0 ;check for hop # rollover
    BLT        OK_2_CONTINUE
    SUBI       *+AR2(NUM_HOP),R0
    STI        R0,@hop_cnt
    LDI        @frm_cnt,R0
    ADDI       1,R0
    CMPI       *+AR2(MAX_FRM),R0 ;check for frm # rollover
    BLE        LEAVE_FRM
    LDI        *+AR2(MAX_FRM),R1
    ADDI       1,R1
    SUBI       R1,R0

LEAVE_FRM:
    STI        R0,@frm_cnt

OK_2_CONTINUE:
    LDI        BITSET,R0
    LDI        DPMEMPG,DP
    STI        R0,@CSYNC_OK

    RETS                        ;return to TMS_UL2.ASM

*****
; State 5: CLR_RESP_PIPE
*****

CLR_PIPE:
    LDI        BSSPG,DP          ;revise current status
    LDI        CLR_RESP_PIPE,R0
    STI        R0,@curr_state

    LDI        DATAPG,DP

```

```

LDI                @RESP_BUF_ADDR,ARO          ;set up indirect addr pointer, ARO

LDI                @last_resp_clr,R0            ;last loc data cleared
LDI                @last_resp_in,R1            ;last loc data entered
CMPI               R0,R1
BEQ                PLINE_MT
STI                R0,@pipe_idx                ;initialize pipeline index

FLUSH_OUT:
LDI                BSSPG,DP                    ;buffer not clear, start emptying
LDI                @pipe_idx,R0                ;update pipeline index
ADDI               1,R0
CMPI               *+AR2(RESP_BUF_SIZ),R0      ;check for pipe index rollover
BLT                SAVE_PIPE_IDX
SUBI               *+AR2(RESP_BUF_SIZ),R0

SAVE_PIPE_IDX:
STI                R0,@pipe_idx

LDI                @pipe_idx,IR0
LDI                *+AR0(IR0),R0              ;look at next response in pipeline
LDI                DATAPG,DP
TSTB               @BITS2_N_3,R0              ;look at chl coarse synch response
BZ                FLUSH_NXT                    ;no "DETECT" in response

;//////////////////////////////////////
; Trigger = DET_REC'D, GO TO STATE 3 (VER_DETECT)
;//////////////////////////////////////

LDI                BSSPG,DP                    ;R0 still has response from buffer
LSH                -16,R0                      ;recover frame # from pipeline buffer
STI                R0,@ref_frame                ; for hypothesis lookup
LDI                @pipe_idx,R1
STI                R1,@last_resp_clr            ;save loc of last response cleared

LDI                DET_REC'D,R0                ;set trigger
STI                R0,@trigger
CALL              CHG_STATE
LDI                BSSPG,DP
LDI                @new_state,R2
B                  R2                          ;go to new state (state 3)
;//////////////////////////////////////

FLUSH_NXT:
LDI                BSSPG,DP
LDI                @pipe_idx,R0                ;find out if any more to flush
CMPI               @last_resp_in,R0            ;pipe_idx - last_resp_in
BNE               FLUSH_OUT                    ;if pointers don't line up, more to flush
;otherwise, pipeline empty

;//////////////////////////////////////
; Trigger = PLINE_CLR, GO TO STATE 2 (GEN_PROBES) TO RESUME PROBING
;//////////////////////////////////////

PLINE_MT:
LDI                BSSPG,DP
LDI                ULGO_BASE,R0                ;reset prb_cmd
STI                R0,@prb_cmd

TRY_NXT_FRM:
LDI                @nxt_prb_frm,R0            ;compute next frame to tx probes

NXT_FRM_2_PRB:
ADDI               4,R0
CMPI               @frm_cnt,R0
BGT               NXT_FRM_2_PRB                ;next frame number is valid
B                  TRY_NXT_FRM                ;need to go to next allocation

NXT_FRM_2_PRB:
CMPI               *+AR2(MAX_FRM),R0           ;check if frame # exceeded
BLE               SAVE_FRM
LDI                *+AR2(MAX_FRM),R1           ;rollover frame
ADDI               1,R1
SUBI               R1,R0

SAVE_FRM:
STI                R0,@nxt_prb_frm
STI                R0,@act_prb_frm
STI                R0,@start_timef

GEN_PROBES
LDI                @last_hyp_tested,R0        ;recover last hypothesis tested in

STI                R0,@hyp_offset
ADDI               @first_prb,R0
STI                R0,@act_prb_hop
CMPI               *+AR2(NUM_HOP),R0
BLT               STORE_HYP
SUBI               *+AR2(NUM_HOP),R0

STORE_HOP:
STI                R0,@act_prb_hop
LDI                @nxt_prb_frm,R0
ADDI               1,R0

```

```

                CMPI            *+AR2(MAX_FRM),R0;check if frame # exceeded
                BLE            FRM_STORE
                LDI            *+AR2(MAX_FRM),R1;rollover frame
                ADDI            1,R1
                SUBI            R1,R0
FRM_STORE:
                STI            R0,@act_prb_frm

STORE_HYP:
                LDI            DATAFG,DP
                LDI            @HYP_LOG_ADDR,AR0
                LDI            BSSPG,DP
                LDI            @nxt_prb_frm,IRO
                LDI            @hyp_offset,R0
                STI            R0,*+AR0(IRO) ;store hyp_offset in hyp_log rel to frame #

                LDI            PLINE_CLR,R0 ;set trigger
                STI            R0,@trigger
                CALL            CHG_STATE
                LDI            BSSPG,DP
                LDI            @new_state,R2
                B                R2 ;go to new state (state 2)
;////////////////////////////////////
*****
; State 6: SRCH_EXCEED
*****
OUT_A_RANGE:
                LDI            BSSPG,DP
                LDI            SRCH_EXCEED,R0
                STI            R0,@curr_state

                LDI            DPMEMPG,DP ;set flag on PC indicating error
                LDI            BITSET,R0
                STI            R0,@RNG_XCDED

                RETS ;return to TMS_UL2.ASM (to replace WAIT_LOOP)

*****
; State 7: PLINE_ERR (set flag to PC)
*****
OFLO_RESP:
                LDI            BSSPG,DP ;update current state
                LDI            PLINE_ERR,R0
                STI            R0,@curr_state

                LDI            DPMEMPG,DP ;set flag on PC
                LDI            BITSET,R0
                STI            R0,@PLINE_FLAG

                RETS ;return to TMS_UL2.ASM

*****
                .globl            CHG_STATE
*****
CHG_STATE:
                PUSH            DP
                PUSH            R0
                PUSHF            R0
                PUSH            R1
                PUSHF            R1
                PUSH            R2
                PUSHF            R2
                PUSH            AR0
                PUSH            IRO

                LDI            BSSPG,DP ;trigger x number of states + curr_state
                LDI            @trigger,R0
                LDI            STATE_COL,R1
                MPYI            R0,R1
                ADDI            @curr_state,R1
                LDI            R1,IRO
                LDI            DATAFG,DP
                LDI            @STAT_ADDR,AR0
                LDI            *+AR0(IRO),R2
                LDI            BSSPG,DP
                STI            R2,@new_state

                POP             IRO
                POP             AR0
                POPF            R2
                POP             R2

```



```

                POPF            R1
                POP            R1
                POPF            R0
                POP            R0
                POP            DP

                RETS

*****
                .globl          INPUT_HYP_LOG
*****

INPUT_HYP_LOG:

                PUSH            DP
                PUSH            IR1
                PUSH            AR0
                PUSH            AR1
                PUSH            AR3
                PUSH            R0
                PUSHF            R0

                LDI            BSSPG,DP
                LDI            @hyp_index,R0
                LDI            R0,IR1

                LDI            DATAPG,DP
                LDI            @HYP_LOG_ADDR,AR0
                LDI            @HYP_FRM_ADDR,AR1
                LDI            *,AR0(IR1),R0 ;only have to check one of them
                CMPI            9999,R0
                BNE            HYP_BUFF_OFLO

                LDI            BSSPG,DP
                LDI            @nxt_prb_frm,R0
                STI            R0,*,AR1(IR1) ;save probe frame number
                LDI            @hyp_offset,R0
                STI            R0,*,AR0(IR1) ;save hypothesis for probe frame number

                ADDI            1,IR1
                CMPI            128,IR1 ;check for hyp_index rollover
                BLT            SAV_HYP_IDX
                LDI            BITCLR,IR1 ;rollover

SAV_HYP_IDX:
                STI            IR1,@hyp_index
                B            END_ROUTINE

HYP_BUFF_OFLO:

                LDI            BITSET,R0
                LDI            DPMEMPG,DP
                STI            R0,@TOO_MANY_HYPS;set flag to PC

END_ROUTINE:

                POPF            R0
                POP            R0
                POP            AR3
                POP            AR1
                POP            AR0
                POP            IR1
                POP            DP

                RETS

*****
                .globl          RETRV_HYP
*****

RETRV_HYP:

                PUSH            DP
                PUSH            IR0
                PUSH            AR0
                PUSH            AR3
                PUSH            R0
                PUSHF            R0

FIND_IDX:
                LDI            DATAPG,DP
                LDI            @HYP_FRM_ADDR,AR0
                LDI            0,IR0

FIND_LOOP:
                LDI            *,AR0(IR0),R0
                LDI            BSSPG,DP
                CMPI            @ref_frame,R0
                BEQ            FOUND_IDX
                ADDI            1,IR0

```

```

        LDI        IR0,R0
        CMPI       128,R0          ;check if at end of buffer
        BLT        FIND_LOOP

        LDI        BITSET,R0      ;set flag to PC, frm not found
        LDI        DPMEMPG,DP
        STI        R0,@FRM_NOT_FOUND

        B          CLOSE_ROUTINE

FOUND_IDX:
        LDI        DATAPG,DP      ;IR0 currently has index
        LDI        @HYP_LOG_ADDR,AR0
        LDI        *+AR0(IR0),R0  ;retrieve hyp_offset
        LDI        BSSPG,DP
        STI        R0,@hyp_offset

        LDI        9999,R0
        STI        R0,*+AR0(IR0)  ;reset hyp_log location
        LDI        DATAPG,DP
        LDI        @HYP_FRM_ADDR,AR0
        STI        R0,*+AR0(IR0)  ;reset hyp_frame location

CLOSE_ROUTINE:
        POPF       R0
        POP        R0
        POP        AR0
        POP        IR0
        POP        DP

        RETS

        .end

```

## B4. Fine synchronization routine

```

*****
*           Program Name:  FSYNC.ASM          *
*           Author:       C. Tom              *
*           Date edited:   31 March 1998      *
* Description: Assembler code to be added to TMS_UL2.ASM which performs *
* fine synchronization. Fine synchronization probes are                *
* transmitted for user 1 (burst of 32). Fine synch responses          *
* are analyzed in PC program. PC will take an average                 *
* of 'X'synch responses and compute an appropriate phase              *
* change for the NCO. The phase change is such that the               *
* adjustment of hop clock over 320 hops (1 frame). After             *
* 320 hops, the NCO is returned to its original frequency.           *
* Fine synchroniztion occurs when fine synch response falls          *
* below a certain threshold (i.e. fsync resp < threshold 'Y')        *
* Currently considering threshold to be within 10% of a hop           *
*****

**** Subroutine declaration ****

                .globl      FINE_SYNC
                .globl      CHG_FSTATE
                .globl      COMMAND_CLK
                .globl      DSPDLAYLP
                .globl      NRDY_low_loop
                .globl      START_NCO

**** Miscellaneous constants ****

XF0_EN          .set        2h
XF_SET          .set        6h
XF_CLR          .set        0FFFBh

BITSET          .set        1
BITCLR          .set        0

MASKL16         .set        0FFFFh

NUM_ROW         .set        6                ;number of rows in state table
NUM_COL         .set        6                ;number of columns in state table

                .data
BITS0_N_1       .word       3h
BITS2_31        .word       0FFFFFFFCh
VAR_BASE        .word       gt_vars
STATE_TBL       .word       STATE_BASE

**** TRIGGERS for FSYNC routine ****

CMD_ISSUED      .set        0                ;starting point of FSYNC routine
INI_COMPL       .set        1                ;finished initialising variables, etc...
F_EST_AVAIL     .set        2                ;fine estimate available from PC
ADJ_COMPL       .set        3                ;adjustment of NCO completed
NT_CONVERGING   .set        4                ;no convergence of estimate after X times
CONVERGED       .set        5                ;fine synch to within 10% of a hop achieved

**** STATES for FSYNC routine ****

IDLE_F          .set        0                ;idle state
INIT_SECTION    .set        1                ;performing preliminary initialisation
TX_FPROBES      .set        2                ;generating fsync probes and waiting for fine est
ADJ_NCO         .set        3                ;adjusting NCO frequency over 320 hops
FINE_NT_ACH     .set        4                ;not able to achieve fine synch, send error msg
GO_2_RUN        .set        5                ;fine sync achieved, ready to send data

**** Look up table for FSYNC routine ****

                ;      trigger
                ;      |      current state
                ;      |      |
                ;      |      |
STATE_BASE:     .data
                .word       FINE_INIT        ;[0,0]=0
                .word       0                ;[0,1]
                .word       0                ;[0,2]
                .word       0                ;[0,3]
                .word       0                ;[0,4]
                .word       0                ;[0,5]
                .word       0                ;[1,0]=6
                .word       FINE_PRB         ;[1,1]
                .word       0                ;[1,2]
                .word       0                ;[1,3]
                .word       0                ;[1,4]
                .word       0                ;[1,5]

```

```

.word 0 ;[2,0]=12
.word 0 ;[2,1]
.word CHANGE_NCO ;[2,2]
.word 0 ;[2,3]
.word 0 ;[2,4]
.word 0 ;[2,5]
.word 0 ;[3,0]=18
.word 0 ;[3,1]
.word 0 ;[3,2]
.word FINE_PRB ;[3,3]
.word NO_CONVERG ;[3,4]
.word FINE_ACH ;[3,5]
.word 0 ;[4,0]=24
.word 0 ;[4,1]
.word 0 ;[4,2]
.word 0 ;[4,3]
.word 0 ;[4,4]
.word 0 ;[4,5]
.word 0 ;[5,0]=30
.word 0 ;[5,1]
.word 0 ;[5,2]
.word 0 ;[5,3]
.word 0 ;[5,4]
.word 0 ;[5,5]

**** Indices for gt_vars parameters array ****

NUM_HOP .set 0
MAX_FRM .set 1
MAX_HOP .set 2
PRB_START .set 3
LIM_10 .set 9
MIN_4_CONV .set 10
MAX_ATTEMPTS .set 11

**** Data page pointers ****

INIPG .set 0
DPMEMPG .set 3h
BSSPG .set 80h
DATAFG .set 0h
DSPLNKPG .set 80h
BUSPG .set 80h
DEBUGPG .set 80h

***** DPMEM addresses ****

DPBASE .set 30000h
FSTART_AVAIL .set DPBASE + 40h
FSTART_FRM .set DPBASE + 41h
FSTART_ACK .set DPBASE + 42h
EST_AVAIL .set DPBASE + 43h
FINE_EST .set DPBASE + 44h
PHS_CHANGE .set DPBASE + 45h
EST_ACK .set DPBASE + 46h
NO_FSYNC .set DPBASE + 47h
FSYNC_OK .set DPBASE + 48h

**** GT i/f board addresses ****

COMMAND .set 800004h ;WRITE only
STATUS .set 800004h ;READ only
NCO_CMD .set 800005h ;WRITE only
INTRPT_PORT .set 800005h ;READ only
HSC_PORT .set 800006h ;WRITE only
FSK_FRM .set 800007h ;WRITE only

**** HSC commands and miscellaneous ****

HSC_endhop .data .word 0fffffh ;16777215 (max hop number for HSC)

SYNC_BIT .set 2 ;SYNC on bit D1 of GT status
NRDY_BIT .set 4 ;bit D2 on GT status (DSPLINK)

STOP_HSC .set 0
RUN_HSC .set 8000h
CHG_IMMED .set 0501h
CHG_HOP .set 0500h
LD_LATCH .set 0300h
LD_BASE .set 030Ch
LD_BWSLACK .set 0306h
LD_DOPF .set 0310h
LD_FCSPACE .set 030Ah
LD_FLAGS .set 031Ah
LD_FSKCHAN .set 0308h
LD_HOP .set 0304h

```

```

LD_LOSCI      .set      0318h
LD_LOCOM      .set      0312h
LD_OFFSET     .set      030Eh
LD_TIMELO     .set      0314h
LD_TIMEHI     .set      0316h

ULSYNC_CMD    .set      500Bh      ;GO TO ULSYNC MODE, channel 1, bin 3
ULGO_BASE     .set      200h      ;BASE VALUE OF ULGO COMMAND FOR CSYNC

U1_FPROBE     .set      13h      ;00010011, FSK/CHAN FOR USER 1
TX_OFF        .set      80h      ;set RF_OFF "high" when not transmitting f probes

**** Reserve memory in .bss for variables ****

        .globl      allocated_f
        .bss        allocated_f,1
        .globl      avg_fine
        .bss        avg_fine,1
        .globl      burst_compl
        .bss        burst_compl,1
        .globl      convrg_cnt
        .bss        convrg_cnt,1
        .globl      no_convrg_cnt
        .bss        no_convrg_cnt,1
        .globl      prev_fine
        .bss        prev_fine,1
        .globl      returned_est
        .bss        returned_est,1
        .globl      tx_fine_en
        .bss        tx_fine_en,1

**** Variables defined elsewhere ****

        .globl      act_prb_frm
        .globl      curr_state
        .globl      frm_cnt
        .globl      frm_ref
        .globl      gt_vars
        .globl      hop_cnt
        .globl      hop_ref
        .globl      new_state
        .globl      nxt_prb_frm
        .globl      original_phase
        .globl      phs_rnded
        .globl      start_timef
        .globl      trigger
        .globl      uflo_err

*****

**** Program begins here ****

*****

        .text

*****
; STATE 0: IDLE_F
*****

FINE_SYNC:
        LDI          BSSPG,DP
        LDI          IDLE_F,R0
        STI          R0,@curr_state

;////////////////////////////////////////
; Trigger = CMD_ISSUED, GO TO STATE 1 (INIT_SECTION)
;////////////////////////////////////////

*****
; STATE 1: INIT_SECTION
*****

FINE_INIT:
        LDI          BSSPG,DP
        LDI          INIT_SECTION,R0
        STI          R0,@curr_state

        LDI          DATAPG,DP
        LDI          @VAR_BASE,AR2

; Step 1: Initialisation of variables, etc...
;         Send command to HSC to go to RUN mode

```

```

        LDI            BSSPG,DP
        LDI            BITCLR,R0
        STI            R0,@burst_compl
        STI            R0,@allocated_f
        STI            R0,@tx_fine_en ;disable transmission of fsync probes
        STI            R0,@no_convrg_cnt
        STI            R0,@convrg_cnt
        STI            R0,@uflo_err

        LDI            31,R0
        STI            R0,@prev_fine ;start max fine estimate

        CALL           NRDY_low_loop ;send RUN command to HSC
        LDI            RUN_HSC,R0
        LSH            16,R0
        LDI            DSPLNKPG,DP
        STI            R0,@HSC_PORT
        CALL           DSPDLAYLP

;////////////////////////////////////
; Trigger = INI_COMPL, GO TO STATE 2 (TX_FPROBES)
;////////////////////////////////////

*****
; STATE 2: TX_FPROBES
*****

FINE_PRB:
        LDI            BSSPG,DP
        LDI            TX_FPROBES,R0
        STI            R0,@curr_state

        LDI            BSSPG,DP ;this is for later iterations
        LDI            BITCLR,R0 ;after estimate is received
        STI            R0,@burst_compl
        STI            R0,@allocated_f

; Step 1: compute next allocation to transmit fine synch probes
; find next multiple of 4 for frame number

        LDI            BSSPG,DP
        LDI            @frm_cnt,R0
        LDI            DATAPG,DP
        TSTB           @BITS0_N_1,R0 ;round to next mult of 4
        BZ             NOW_A_MULT_4 ;if zero, already a mult of 4
ADV_NXT_4:
        LDI            DATAPG,DP
        AND            @BITS2_31,R0 ;mask off 2 LSBs
                                           ;DON'T KNOW IF NEED TO ADD ANOTHER 4!!!
NOW_A_MULT_4:
        ADDI           4,R0
        CMPI           *+AR2(MAX_FRM),R0;check if # exceeds max frame
        BLE            BLOW_MAXF
ABOV_MAXF:
        LDI            *+AR2(MAX_FRM),R1
        ADDI           1,R1
        SUBI           R1,R0
BLOW_MAXF:
        LDI            BSSPG,DP
        STI            R0,@start_timef ;HAVE TO SEND TO PC SO IT KNOWS WHEN
        STI            R0,@nxt_prb_frm ;TO START ANALYZING SYNCH RESPONSES
        STI            R0,@act_prb_frm

; Step 2: transfer start_timef to PC

        LDI            DPMEMPG,DP
        STI            R0,@FSTART_FRM ;R0 already has start_timef

        LDI            BITSET,R0
        STI            R0,@FSTART_AVAIL
FSTART_WAIT:
        LDI            @FSTART_ACK,R0
        CMPI           BITSET,R0
        BNE            FSTART_WAIT

        LDI            BITSET,R0
        LDI            BSSPG,DP
        STI            R0,@tx_fine_en ;enable transmission of fsync probes

; Step 3: initialise/set flags for ISR, including read status to clear uflo bit
CONT_W_FPRBS:
        LDI            BSSPG,DP
        LDI            @uflo_err,R0 ;check for data underflow condition
        CMPI           BITSET,R0
        BEQ            DATA_ERROR

```

```

LDI      BSSPG,DP
LDI      @burst_compl,R0 ;check if burst is completed
CMPI     BITSET,R0
BNE      TIME_TRANSF

LDI      BITCLR,R0
STI      R0,@burst_compl ;reset variables for next burst
STI      R0,@allocated_f

LDI      DSPLNKPG,DP ;NT SURE IF NEEDED,TX_OFF SENT WHEN allocated_f=0
LDI      @STATUS,R0 ;read GT status to clear uflo bit

LDI      BSSPG,DP
LDI      @nxt_prb_frm,R0
ADDI     4,R0
CMPI     **AR2(MAX_FRM),R0;check for rollover of frame #
BLE      FRM_AS_IS
LDI      **AR2(MAX_FRM),R1
ADDI     1,R1
SUBI     R1,R0
FRM_AS_IS:
STI      R0,@nxt_prb_frm ;save nxt frame # for fsync probes
STI      R0,@act_prb_frm

; Step 4: check if time to transmit probes

TIME_TRANSF:
LDI      BSSPG,DP
LDI      @allocated_f,R0 ;check if transmitting a burst already
CMPI     BITSET,R0
BEQ      CHK_4_F_EST

LDI      @act_prb_frm,R0
LDI      @frm_cnt,R1 ;if not transmitting a burst,
CMPI     R0,R1 ;check if it's an allocated frame
BNE      CHK_4_F_EST
SET_FPRB_FLAG:
LDI      BITSET,R0
STI      R0,@allocated_f

CHK_4_F_EST:
LDI      DPMPMPG,DP
LDI      @EST_AVAIL,R0 ;check if fine estimate is available
CMPI     BITSET,R0
BNE      CONT_W_FPRBS

LDI      DPMPMPG,DP
LDI      BITCLR,R0
STI      R0,@EST_AVAIL
LDI      @FINE_EST,R0 ;read estimate from PC (avg of 10 responses)
LDI      @PHS_CHANGE,R1 ;read phase change calc by PC
LDI      BSSPG,DP
STI      R0,@avg_fine ;avg_fine = average of fine est received
STI      R1,@returned_est ;returned_est = new NCO phase for adj

LDI      BITSET,R0
LDI      DPMPMPG,DP
STI      R0,@EST_ACK ;ack estimate received to PC

////////////////////////////////////
; Trigger = F_EST_AVAIL, GO TO STATE 3 (ADJ_NCO)
////////////////////////////////////

*****
; STATE 3: ADJ_NCO
*****

LDI      BSSPG,DP
LDI      ADJ_NCO,R0
STI      R0,@curr_state

; Step 1: check for convergence or non-convergence of fine synch estimate

LDI      BSSPG,DP
LDI      @avg_fine,R0
ABSI     R0,R1
CMPI     **AR2(LIM_10),R1 ;check if within 10% of hop period
BLE      INCR_CNVRG
CMPI     @prev_fine,R1 ;check if estimate is still converging
BGT      INCR_NCNVRG ;no -> update non-convergence count
STI      R1,@prev_fine ;yes -> update prev_fine (=abs(estimate))
B        INCR_CNVRG ; update convergence count
INCR_NCNVRG:
LDI      BSSPG,DP
LDI      @no_converg_cnt,R0;update non-convergence count
ADDI     1,R0
STI      R0,@no_converg_cnt

```

```

                ADDI        @convrg_cnt,R0
                CMPI        *+AR2(MAX_ATTEMPTS),R0    ;check max attempts desired exceeded?
                BLT         CHANGE_NCO

;//////////////////////////////////////
; Trigger = NT_CONVERGING, GO TO STATE 4 (FINE_NT_ACH)
;//////////////////////////////////////

                LDI         BSSPG,DP
                LDI         NT_CONVERGING,R0
                STI         R0,@trigger
                CALL        CHG_FSTATE
                LDI         BSSPG,DP
                LDI         @new_state,R2
                B           R2

INCR_CNVRG:
                LDI         BSSPG,DP
                LDI         @convrg_cnt,R0    ;update convergence count
                ADDI        1,R0
                CMPI        *+AR2(MIN_4_CONV),R0    ;check if converged enough times
                BLT         CHANGE_NCO

;//////////////////////////////////////
; Trigger = CONVERGED, GO TO STATE 5 (GO_2_RUN)
;//////////////////////////////////////

                LDI         BSSPG,DP
                LDI         CONVERGED,R0
                STI         R0,@trigger
                CALL        CHG_FSTATE
                LDI         BSSPG,DP
                LDI         @new_state,R2
                B           R2

CHANGE_NCO:
                LDI         BITCLR,R0
                STI         R0,@tx_fine_en    ;stop fine probes during NCO adjustment

                LDI         BSSPG,DP
                LDI         @returned_est,R0
                STI         R0,@phs_rnded

                CALL        START_NCO    ;change phase value on NCO

                LDI         BSSPG,DP
                LDI         @hop_cnt,R0
                LDI         @frm_cnt,R1
                STI         R0,@hop_ref    ;save current hop and frame #
                STI         R0,@frm_ref

WAIT_1_FRM:
                LDI         BSSPG,DP    ;NCO adjusted over 320 hops
                LDI         @hop_cnt,R0
                CMPI        @hop_ref,R0
                BNE         WAIT_1_FRM
                LDI         @frm_cnt,R0
                CMPI        @frm_ref,R0
                BEQ         WAIT_1_FRM

                LDI         BSSPG,DP
                LDI         @original_phase,R0
                STI         R0,@phs_rnded

                CALL        START_NCO    ;change NCO phase back to orig value

;//////////////////////////////////////
; Trigger = ADJ_COMPL, GO TO STATE 2 (TX_FPROBES)
;//////////////////////////////////////

                LDI         BSSPG,DP
                LDI         ADJ_COMPL,R0
                STI         R0,@trigger
                CALL        CHG_FSTATE
                LDI         BSSPG,DP
                LDI         @new_state,R2
                B           R2

*****
; STATE 4: FINE_NT_ACH
*****

NO_CONVERG:
                LDI         BSSPG,DP
                LDI         FINE_NT_ACH,R0
                STI         R0,@curr_state

```



```

        LDI        BITSET,R0        ;set flag on PC
        LDI        DPMEMPG,DP
        STI        R0,@NO_FSYNC

        RETS                        ;return to TMS_UL2.ASM

*****
; STATE 5: GO_2_RUN
*****

FINE_ACH:
        LDI        BSSPG,DP
        LDI        GO_2_RUN,R0
        STI        R0,@curr_state

        LDI        DPMEMPG,DP        ;set flag on PC
        LDI        BITSET,R0
        STI        R0,@FSYNC_OK

        RETS                        ;return to TMS_UL2.ASM

*****
; DATA_ERROR STATE, return to TMS_UL2.ASM, flag to PC already set in ISR
*****

DATA_ERROR:
        RETS

*****
        .globl      CHG_FSTATE
*****

CHG_FSTATE:
        PUSH        DP
        PUSH        R0
        PUSHF       R0
        PUSH        R1
        PUSHF       R1
        PUSH        R2
        PUSHF       R2
        PUSH        AR0
        PUSH        IR0

        LDI        BSSPG,DP
        LDI        @trigger,R0
        LDI        NUM_COL,R1
        MPYI        R0,R1
        ADDI        @curr_state,R1
        LDI        R1,IR0
        LDI        DATAPG,DP
        LDI        @STATE_TBL,AR0
        LDI        *+AR0(IR0),R2
        LDI        BSSPG,DP
        STI        R2,@new_state

        POP         IR0
        POP         AR0
        POPF        R2
        POP         R2
        POPF        R1
        POP         R1
        POPF        R0
        POP         R0
        POP         DP

        .end

```

## B5. DSP interrupt service routine

```

*****
*                               Program Name:  UL_ISR.ASM                               *
*                               Author:       C. Tom                               *
*                               Date:        09 May 1997                               *
*                               Edited:      30 March 1998                               *
* Description:  Interrupt service routine for uplink synchronization for             *
*               the ground terminal simulator.  On each rising edge of hop           *
*               clock, GT status is read, hop/frn counters are updated,             *
*               able to check for FR0 pulse, and check whether to send              *
*               coarse synch probes                                                 *
*****

**** Subroutine declarations ****

                .globl          GT_ISR

**** Miscellaneous constants ****

XF0_EN          .set           2h
XF_SET          .set           6h
XF_CLR          .set           0FFFBh

BITCLR          .set           0
BITSET          .set           1

MODE4           .set           4           ;detect FR0 enable mode
MODE6           .set           6           ;coarse synch mode
MODE7           .set           7           ;fine synch mode

VAR_BASE        .data
                .word          gt_vars

* Indices for gt_vars.parameters array

NUM_HOP         .set           0
MAX_FRM         .set           1
MAX_HOP         .set           2
PRB_START       .set           3
SRCH_LIM        .set           4
TIMES_4_CONFM   .set           5
RESP_BUF_SIZ    .set           6

**** Status bits mask ****

BER_tx_rdy      .set           8000h      ;BIT 15 of BER status
BER_oflo_bit    .set           4000h      ;BIT 14 of BER status
FR0_BIT         .set           800h       ;BIT 11 of BER status
HCLK_BIT        .set           1h         ;BIT 0 OF GT status
DAT_UFLO        .set           8h         ;BIT 3 OF GT status

                .data
PN_LIMIT        .word          0FFFFFFh   ;HSC max hop number 16777215
DBGU_ADDR       .word          809A00h     ;addr of first entry in dbug array

**** Data page pointers ****

INIPG           .set           0h
DPMEMPG         .set           3h
BSSPG          .set           80h
DATAPG          .set           0h
DSPLNKPG       .set           80h
BUSPG          .set           80h
DBGUPG         .set           80h

**** DPMEM addresses ****

* DPBASE to DPBASE+17h, DPBASE+30h TO DPBASE+34h used in TMS_UL2.ASM
* DPBASE+19h to DPBASE+2Bh, DPBASE+2Eh used in CSYNC.ASM

RIS_DET         .set           30018h
CHK_HOP         .set           3002Ch
CHK_FRM         .set           3002Dh
UFLO_CDTN       .set           30049h

**** BER i/f board addresses ****

CMD_BER         .set           800009h     ;WRITE only
STAT_BER        .set           800009h     ;READ only
BER_DAT_PORT    .set           800008h     ;READ/WRITE

**** GT i/f board addresses ****

```

```

COMMAND      .set      800004h      ;WRITE only
STATUS       .set      800004h      ;READ only
INTRPT_PORT  .set      800005h      ;READ only
HSC_PORT     .set      800006h      ;WRITE only
FSK_FRM      .set      800007h      ;WRITE only

**** HSC commands ****

FCALC        .set      5800h
UI_FPROBE    .set      13h          ;00010011, FSK/CHAN FOR USER 1
TX_OFF       .set      80h          ;set RF_OFF "high" when not transmitting f probes

**** Reserve memory in .bss for variables ****

        .globl      BER_stat
        .bss        BER_stat,1
        .globl      hop_cnt
        .bss        hop_cnt,1
        .globl      frm_cnt
        .bss        frm_cnt,1
        .globl      BER_data
        .bss        BER_data,1
        .globl      prev_FR0
        .bss        prev_FR0,1
        .globl      chk_FR0_flg
        .bss        chk_FR0_flg,1
        .globl      array_cnt
        .bss        array_cnt,1
        .globl      fburst_cnt
        .bss        fburst_cnt,1

**** Variables defined elsewhere ****

        .globl      op_mode
        .globl      tms_csync_rdy
        .globl      assigned_f
        .globl      brst0_flg
        .globl      brst1_flg
        .globl      hyp_used
        .globl      act_prb_hop
        .globl      prb_cmd
        .globl      debug_adr
        .globl      gt_vars
        .globl      allocated_f
        .globl      burst_compl
        .globl      tx_fine_en

        .text

GT_ISR:

        PUSH        DP
        PUSH        ST
        PUSH        IE
        PUSH        IOF
        PUSH        R0
        PUSHF       R0
        PUSHF       R1
        PUSHF       R2
        PUSHF       R2
        PUSH        AR0
        PUSH        IR0
        PUSH        AR1
        PUSH        IR1
        PUSH        AR2

;Step 1: Read GT i/f board interrupt port to clear interrupt

        LDI         DSPLNKP,DP
        LDI         @INTRPT_PORT,R0 ;read GT INTRPT_PORT to clear interrupt
        NOP
        NOP
        NOP

;Step 2: Update hop and frame counters

        LDI         DATAPG,DP
        LDI         @VAR_BASE,AR2

        LDI         BSSPG,DP
        LDI         @hop_cnt,R0 ;update hop and frame counters
        CMPI        *,AR2(MAX_HOP),R0
        BLT         INCR_BY_1
        LDI         0,R0
        STI         R0,@hop_cnt
        LDI         @frm_cnt,R0

```

```

                CMPI      *+AR2(MAX_FRM),R0
                BLT       INCR_FRM

RES_FRM_CNT:
                LDI       0,R0
                LDI       BSSPG,DP
                STI       R0,@frm_cnt
                B         WHICH_MODE

INCR_FRM:
                ADDI      1,R0
                STI       R0,@frm_cnt
                B         WHICH_MODE

INCR_BY_1:
                ADDI      1,R0
                STI       R0,@hop_cnt

;Step 3: Determine which mode of operation GT sync processor is in

WHICH_MODE:
                LDI       BSSPG,DP
                LDI       @op_mode,R0
                CMPI      MODE6,R0
                BEQ       COARSE
                CMPI      MODE7,R0
                BEQ       FINE
                CMPI      MODE4,R0
                BEQ       DET_FRO
                B         WRAP_UP ;do nothing else for this interrupt

*****
; Code for Coarse Synch mode, probe transmission

COARSE:
                LDI       BSSPG,DP
                LDI       @tms_csync_rdy,R0; check if precalc time has elapsed
                CMPI      BITSET,R0
                BNE       WRAP_UP

                LDI       @hop_cnt,R0 ;check for hop = 0
                CMPI      0,R0
                BNE       FR_ASSIGN_CHK

                LDI       DSPLNKP,DP ;issue FCALC command if hop = 0
                LDI       FCALC,R0
                LSH       16,R0
                STI       R0,@HSC_PORT

FR_ASSIGN_CHK:
                LDI       BSSPG,DP
                LDI       @assigned_f,R0 ;is current frame assigned
                CMPI      BITSET,R0 ; csynch frame
                BNE       WRAP_UP

; for now, issue both burst 0 and burst 1 probes from ISR, i.e., wait
; falling edge in ISR to send burst 1 probes...not as efficient...
; will see later about a hardware fix to switch to /HCLK

                LDI       BSSPG,DP
                LDI       @brst0_flg,R0 ;check if already started probe tx
                CMPI      BITSET,R0
                BEQ       TX_B0

                LDI       @brst1_flg,R0 ;check if already started probe tx
                CMPI      BITSET,R0
                BEQ       TX_B1

HOP_CHK:
                LDI       BSSPG,DP
                LDI       @act_prb_hop,R0
                LDI       @hop_cnt,R1
                CMPI      R0,R1 ;(R1-R0) -> R1
                BLT       WRAP_UP ;do nothing if hop_cnt < act_prb_hop
                LDI       BITSET,R0
                STI       R0,@brst0_flg

TX_B0:
                LDI       BSSPG,DP
                LDI       @prb_cmd,R0
                LSH       16,R0
                LDI       DSPLNKP,DP
                STI       R0,@HSC_PORT ;send ULGO command, sync probe
                LSH       -16,R0
                ADDI      2,R0
                LDI       BSSPG,DP

```

```

        STI        R0,@prb_cmd      ;update ULGO command
        CMPI       220h,R0          ;check if all of brst 0 probes sent
        BNE        WRAP_UP
        LDI        BITCLR,R0
        LDI        BSSPG,DP
        STI        R0,@brst0_flg    ;clear brst0_flg
        LDI        BITSET,R0
        STI        R0,@brst1_flg    ;set brst1_flg
        B          WRAP_UP

TX_B1:
WAIT_FALL_HCLK:
        LDI        DSPLNKPG,DP
        LDI        @STATUS,R0      ;wait for falling edge of HCLK
        LSH        -16,R0
        TSTB       HCLK_BIT,R0
        BNZ        WAIT_FALL_HCLK

        LDI        BSSPG,DP
        LDI        @prb_cmd,R0
        LSH        16,R0
        LDI        DSPLNKPG,DP
        STI        R0,@HSC_PORT    ;send ULGO command, sync probe
        LSH        -16,R0
        ADDI       2,R0
        LDI        BSSPG,DP
        STI        R0,@prb_cmd      ;update ULGO command
        CMPI       240h,R0          ;check if all of brst 1 probes sent
        BNE        WRAP_UP
        LDI        BITSET,R0
        LDI        BSSPG,DP
        STI        R0,@hyp_used     ;signal TMS that hyp sent
        B          WRAP_UP

*****
;Code for fine synch mode, transmitting fine synch probes or RF_OFF

FINE:
        LDI        DSPLNKPG,DP
        LDI        @STATUS,R0
        LSH        -16,R0
        TSTB       DAT_UFLO,R0
        BZ         FLO_OK

        LDI        BITSET,R0        ;THIS IS NOT A CLEAN EXIT
        LDI        DPMEMPG,DP
        STI        R0,@UFLO_CDTN

        B          WRAP_UP

FLO_OK:
        LDI        BSSPG,DP
        LDI        @tx_fine_en,R0
        CMPI       BITSET,R0
        BNE        SEND_RF_OFF      ;RF_OFF sent every hop as default

        LDI        @allocated_f,R0
        CMPI       BITSET,R0
        BNE        SEND_RF_OFF      ;RF_OFF sent every hop as default

        LDI        @hop_cnt,R0
        CMPI       **AR2(PRB_START),R0 ;check for hop >= "288"
        BLT        SEND_RF_OFF      ;RF_OFF sent every hop as default

        LDI        U1_FPROBE,R0
        LSH        16,R0
        LDI        DSPLNKPG,DP
        STI        R0,@FSK_FRM      ;send user 1 fine synch probe

        LDI        BSSPG,DP
        LDI        @fburst_cnt,R0
        ADDI       1,R0
        STI        R0,@fburst_cnt
        CMPI       32,R0            ;check if all probes sent yet
        BLT        WRAP_UP

        LDI        BITSET,R0
        STI        R0,@burst_compl  ;set flag for TMS that burst has been sent

        B          WRAP_UP

SEND_RF_OFF:
        LDI        TX_OFF,R0
        LSH        16,R0
        LDI        DSPLNKPG,DP
        STI        R0,@FSK_FRM      ;default command to FSK/CHAN port

        B          WRAP_UP

```

\*\*\*\*\*

;Code for DL synch mode, detecting FR0 pulse and adjusting hop/frm coutners

```

DET_FR0:
    LDI        BSSPG,DP
    LDI        @chk_FR0_flg,R0
    CMPI       BITSET,R0
    BNE        WRAP_UP

CHK_BER_STAT:
    LDI        DSPLNKPG,DP
    LDI        @STAT_BER,R0        ;read BER status
    NOP
    NOP
    NOP
    LSH        -16,R0
    LDI        BSSPG,DP
    STI        R0,@BER_stat

GET_FR0:
    LDI        BSSPG,DP
    LDI        @BER_stat,R0
    AND        FR0_BIT,R0
    CMPI       @prev_FR0,R0
    BEQ        WRAP_UP
    BGT        SYNC_RIS

SYNC_FALL:
    LDI        BSSPG,DP
    STI        R0,@prev_FR0        ;falling edge detected, hop0,fr0
    LDI        0,R0
    STI        R0,@hop_cnt
    ADDI       1,R0
    STI        R0,@frm_cnt
    B          WRAP_UP

SYNC_RIS:
    LDI        BSSPG,DP
    STI        R0,@prev_FR0        ;rising edge detected, hop0,fr0
    LDI        @hop_cnt,R0        ;transfer hop and frame count before reset
    LDI        @frm_cnt,R1
    DPMEMPG,DP
    STI        R0,@CHK_HOP
    STI        R1,@CHK_FRM
    LDI        BSSPG,DP

RESET_CNTS:
    LDI        0,R0
    STI        R0,@hop_cnt
    STI        R0,@frm_cnt

    LDI        DPMEMPG,DP
    LDI        BITSET,R0
    STI        R0,@RIS_DET        ;signal PC that rising edge detected

WRAP_UP:
    LDI        0,IF
    POP        AR2
    POP        IR1
    POP        AR1
    POP        IR0
    POP        AR0
    POPF       R2
    POP        R2
    POPF       R1
    POP        R1
    POPF       R0
    POP        R0
    POP        IOF
    POP        IE
    POP        ST
    POP        DP

    RETI

    .end

```

## B6. TMS Linker .cmd File

```
/*.....*/
/*          Program Name: UL2.CMD          */
/*          Date: 06 October 1997          */
/*          */
/* Previously, you had to enter:            */
/*          lnk30 -c <input filenames> -o <output filename> -l rts.lib          */
/*          */
/*.....*/

/* Input filenames */

tms_ul2.obj intrpts.obj ul_isr.obj csync.obj fsync.obj

-m tms_ul2.map          /* Create a map file */
-o tms_ul2.out          /* Output filename */

/* Specify memory map for c30 */

MEMORY
(
    VECS:      origin = 0          length = 0x40      /* Int vectors */
    Prg_mem:   origin = 0x40       length = 0x9fc0     /* Program memory */
    DP_mem:    origin = 0x30000    length = 0x10000    /* Dual port memory */
    RAM_0:     origin = 0x809800   length = 0x400     /* RAM Block 0 */
    RAM_1:     origin = 0x809c00   length = 0x400     /* RAM Block 1 */
    EXT_MEM0:  origin = 0x80A000   length = 0xC00     /* Ext mem block 0 */
    EXT_MEM1:  origin = 0x80AC00   length = 0xC00     /* Ext mem block 1 */
)

/* Specify "Sections" allocations into memory */

SECTIONS
(
    VECTORS      0000000h      : {}
    .text        : {} > Prg_mem      /* Code */
    .cinit        : {} > Prg_mem      /* Initialization tables */
    .stack        : {} > RAM_0        /* System stack */
    .data         : {} > Prg_mem      /* Assign memory for .data section */
    .bss          : {} > RAM_1        /* Global & static variables */
    .sysmem       : {} > RAM_0        /* Dynamic memory */
    DEBUG_LOG     : {} > EXT_MEM0     /* Debug log area */
)
```

## Appendix C: ASCII Data Files

### C1. General

The three ASCII data files that contain parameters required to operate the GT simulator are listed in this appendix. The use of ASCII data files facilitates changes to parameters without having to recompile the assembly and C programs. The three data files are: "freq.dat", "hscinit.dat", and "Gtparam.dat".

### C2. Freq.dat file

```
\ ASCII file of frequency parameters - "freq.dat"
\ Created: 19 March 1997
\
\ Ensure indices correspond to PC program indices for frequency values
\
\      index      value(hex)      description
\      -----
\
\      0      0X3B9ACA00L      LO_BAND 1 GHz for COMSTRON, in Hz
\      1      0X77359400L      UP_BAND 2 GHz for COMSTRON, in Hz
\      2      0X59682F00L      MID_BAND 1.5 GHz for COMSTRON, in Hz
\      3      0X4A817C80L      ONEQ_BAND 1.25 GHz for COMSTRON, in Hz
\      4      0X684EE180L      THREEQ_BAND 1.75 GHz for COMSTRON, in Hz
d      0      0X1A96F220L      LO_BAND 44.61 GHz /100 Hz for SCITEQ
d      1      0X1A9CE590L      UP_BAND 44.649 GHz /100 Hz for SCITEQ
d      2      0X1A99EBD8L      MID_BAND 44.6295 GHz/100 Hz for SCITEQ
d      3      0X1A986EFCL      ONEQ_BAND 44.61975 GHz/100 Hz for SCITEQ
d      4      0X1A9B68B4L      THREEQ_BAND 44.63925 GHz/100 Hz for SCITEQ
d      5      0x1234L          STOP_FSEL
\      6      0X3B9ACA00L      Base frequency 1 GHz for COMSTRON, in Hz
\      7      0X3B9ACA00L      Hop BW 1 GHz for COMSTRON, in Hz
d      6      0X1A96F220L      Base frequency 44.61 GHz/100 Hz for SCITEQ
d      7      0X5F370L        Hop BW 39 MHz/100 Hz for SCITEQ
\
\ End of file
```



### C3. Hscinit.dat file

```

\ ASCII file of HSC parameters - "hscinit.dat"
\ Created: 5 March 1997
\
\ Ensure indices correspond to TMS program indices for HSC parameters
\
\ VALUES FOR THE SCITEQ
\
\      index      value(hex)      description ( < 40 char)
\      -----
d      0          0F220          BASE_L16 44.61GHz/100Hz
d      1          1A96          BASE_H16 44.61GHz/100Hz
d      2          0F9B8          BWSCALE_L16 39MHz/2/100Hz
d      3          2            BWSCALE_H16 39MHz/2/100Hz
d      4          17C0          HOP_BW_L16 39MHz
d      5          253          HOP_BW_H16 39MHz
d      6          0            DOPF_L16
d      7          0            DOPF_H16
d      8          0            FCSPACE_L16 no FSK modulation
d      9          0            FCSPACE_H16 no FSK modulation
d     10          0            FLAGS_L16
d     11          0            FLAGS_H16
d     12          1060          LOSCI_L16 (BASE - 1.5GHz)/100Hz Ray
d     13          19B2          LOSCI_H16 (BASE - 1.5GHz)/100Hz Ray
d     14          1060          LOCOM_L16 (BASE - 1.5GHz)/100Hz Ray
d     15          19B2          LOCOM_H16 (BASE - 1.5GHz)/100Hz Ray
d     16          0            OFFSET_L16
d     17          0            OFFSET_H16
d     18          5A81          TIMELO_L16 hop129,frame72 (arbitrary)
d     19          0            TIMELO_H16
d     20          0            TIMEHI_L16
d     21          0            TIMEHI_H16
\
\ VALUES FOR THE COMSTRON
\
\      index      value(hex)      description ( < 40 char)
\      -----
\      0          9680          BASE_L16 1GHz/100Hz Comstron
\      1          98          BASE_H16 1GHz/100Hz Comstron
\      2          4B40          BWSCALE_L16 1GHz/2/100Hz Comstron
\      3          4C          BWSCALE_H16 1GHz/2/100Hz Comstron
\      4          0CA00          HOP_BW_L16 1GHz Comstron
\      5          369A          HOP_BW_H16 1GHz Comstron
\      6          0            DOPF_L16
\      7          0            DOPF_H16
\      8          0            FCSPACE_L16 no FSK modulation
\      9          0            FCSPACE_H16 no FSK modulation
\     10          0            FLAGS_L16
\     11          0            FLAGS_H16
\     12          0            LOSCI_L16 Comstron
\     13          0            LOSCI_H16 Comstron
\     14          0            LOCOM_L16 Comstron
\     15          0            LOCOM_H16 Comstron
\     16          0            OFFSET_L16
\     17          0            OFFSET_H16
\     18          5A81          TIMELO_L16 hop129,frame72 (arbitrary)
\     19          0            TIMELO_H16
\     20          0            TIMEHI_L16
\     21          0            TIMEHI_H16
\
\End of file

```

#### C4. Gtparam.dat file

\Data file for parameters to be downloaded to TMS memory for GT processor

\ "Gtparam.dat"

\ Created: 19 March 1997

\

\Type	Index	Value	Comments
----	-----	-----	-----
\	0	320	NUM_HOP = number of hops
d	1	191	MAX_FRM = maximum frame number
d	2	319	MAX_HOP = maximum hop number
d	3	288	PRB_START = starting hop number for probes
d	4	32	SRCH_LIM = search limit for hypothesis
d	5	5	TIMES_4_CONFM = # detects to confirm coarse synch
d	6	25	RESP_BUF_SIZ = synch resp buffer size
d	7	5	MIN_DET_2_VER = min # of detects before verified
d	8	15	NUM_RETRANSMITS = # of attempts to verify detect
d	9	6	LIM_10 = est range within 10 % of a hop
d	10	5	MIN_4_CONV = # times f est in LIM_10 range
d	11	25	MAX_ATTEMPTS = max # of attempts to adjust NCO

\

\End of file

## UNCLASSIFIED

SECURITY CLASSIFICATION OF FORM  
(highest classification of Title, Abstract, Keywords)

## DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

<b>1. ORIGINATOR</b> (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Establishment sponsoring a contractor's report, or tasking agency, are entered in section 8.) Defence Research Establishment Ottawa Ottawa, Ontario K1A 0Z4		<b>2. SECURITY CLASSIFICATION</b> (overall security classification of the document including special warning terms if applicable)  <b>UNCLASSIFIED</b>	
<b>3. TITLE</b> (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C or U) in parentheses after the title.)  Ground Terminal Simulator Implementation for Uplink Synchronization Trials (U)			
<b>4. AUTHORS</b> (Last name, first name, middle initial) Tom, Caroline			
<b>5. DATE OF PUBLICATION</b> (month and year of publication of document) November 1998		<b>6a. NO. OF PAGES</b> (total containing information. Include Annexes, Appendices, etc.) 129	<b>6b. NO. OF REFS</b> (total cited in document) 11
<b>7. DESCRIPTIVE NOTES</b> (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.) DREO Report			
<b>8. SPONSORING ACTIVITY</b> (the name of the department project office or laboratory sponsoring the research and development. Include the address.) 5CA11			
<b>9a. PROJECT OR GRANT NO.</b> (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant)		<b>9b. CONTRACT NO.</b> (if appropriate, the applicable number under which the document was written)	
<b>10a. ORIGINATOR'S DOCUMENT NUMBER</b> (the official document number by which the document is identified by the originating activity. This number must be unique to this document.) DREO REPORT 1341		<b>10b. OTHER DOCUMENT NOS.</b> (Any other numbers which may be assigned this document either by the originator or by the sponsor)	
<b>11. DOCUMENT AVAILABILITY</b> (any limitations on further dissemination of the document, other than those imposed by security classification) <input checked="" type="checkbox"/> Unlimited distribution <input type="checkbox"/> Distribution limited to defence departments and defence contractors; further distribution only as approved <input type="checkbox"/> Distribution limited to defence departments and Canadian defence contractors; further distribution only as approved <input type="checkbox"/> Distribution limited to government departments and agencies; further distribution only as approved <input type="checkbox"/> Distribution limited to defence departments; further distribution only as approved <input type="checkbox"/> Other (please specify):			
<b>12. DOCUMENT ANNOUNCEMENT</b> (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). however, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.) Unlimited Announcement			

UNCLASSIFIED

SECURITY CLASSIFICATION OF FORM

RA.W (21 Dec 92)

---

**UNCLASSIFIED**

---

SECURITY CLASSIFICATION OF FORM

13. **ABSTRACT** (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual).

A ground terminal (GT) simulator was developed at Defence Research Establishment Ottawa (DREO) as part of an in-house activity examining aspects of uplink synchronization for extremely high frequency (EHF) satellite communications (SATCOM) using frequency hopping. The GT simulator consists of a GT processor, custom interface boards, synthesizer controller, frequency synthesizer, and data source. The GT processor is the principal component of the simulator and is realized by a TMS320C30 digital signal processor board. This report describes the implementation of the GT processor functions relating to uplink synchronization and the interfaces between the various components of the simulator. This report also describes the synchronization procedure for the GT simulator. The procedure is broken down into three steps: downlink synchronization; uplink coarse synchronization; and uplink fine synchronization. A guide on the hardware installation of the various components of the GT simulator and a list of the executable files needed to run the simulator is provided in an appendix.

14. **KEYWORDS, DESCRIPTORS or IDENTIFIERS** (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

ground terminal simulator  
uplink synchronization  
coarse synchronization  
fine synchronization  
digital signal processor  
EHF satcom

---

**UNCLASSIFIED**

---

SECURITY CLASSIFICATION OF FORM